

Greedy Strategies and Larger Islands of Tractability for Conjunctive Queries and Constraint Satisfaction Problems

Gianluigi Greco^a, Francesco Scarcello^b

^a*Dipartimento di Matematica e Informatica, Università della Calabria, I-87036 Rende(CS), Italy*

^b*DIMES, Università della Calabria, I-87036 Rende(CS), Italy*

Abstract

Structural decomposition methods have been developed for identifying tractable classes of instances of fundamental problems in databases, such as conjunctive queries and query containment, of the constraint satisfaction problem in artificial intelligence, or more generally of the homomorphism problem over relational structures. These methods work on the hypergraph structure of problem instances. Each method provides a way of transforming any cyclic hypergraph into an acyclic one, by organizing its edges (or its nodes) into a polynomial number of clusters, and by suitably arranging these clusters as a tree, called decomposition tree. Then, by using such a tree (or by just knowing that any exists) the given problem instance can be solved in polynomial time.

Most structural decomposition methods can be characterized through hypergraph games that are variations of the *Robber and Cops* graph game that characterizes the notion of treewidth. In particular, decomposition trees somehow correspond to *monotone winning strategies*, where the escape space of the robber on the hypergraph is shrunk monotonically by the cops. In fact, unlike the treewidth case, there are hypergraphs where monotonic strategies do not exist, while the robber can be captured by means of more complex non-monotonic strategies. However, these powerful strategies do not correspond in general to valid decompositions.

The paper provides a general way to exploit the power of non-monotonic strategies, by allowing a “disciplined” form of non-monotonicity, characteristic of cops playing in a *greedy* way. It is shown that deciding the existence of a (non-monotone) greedy winning strategy (and compute one, if any) is tractable. Moreover, despite their non-monotonicity, such strategies always induce valid decomposition trees, which can be computed efficiently based on them. As a consequence, greedy strategies allow us to define new islands of tractability for the considered problems, properly including all previously known classes of tractable instances. In particular, we define the new notion of greedy hypertree decomposition of a hypergraph, whose associated notion of width is at most the hypertree width, and sometimes strictly smaller.

Keywords: Structural Decomposition Methods, Games on Discrete Structures, Conjunctive Queries and Databases, Constraint Satisfaction Problems, Hypertree Decompositions, Tree Projections, Homomorphism Problem

Email addresses: ggreco@mat.unical.it (Gianluigi Greco), scarcello@dimes.unical.it

1. Introduction

We look for islands of tractability for answering conjunctive queries over relational databases or, equivalently, for solving constraint satisfaction problems. For the sake of presentation, we next focus on the database setting and the conjunctive query answering problem. We remark that all results can immediately be applied to all problems that can be recast as homomorphism problems, and possibly can be useful in further settings, thanks to the general combinatorial nature of the proposed approach. We refer the interested reader to [31] for more detail on the connections with the homomorphism problem and with further equivalent problems.

1.1. Acyclic Conjunctive Queries

Conjunctive queries are defined through conjunctions of atoms (without negation), and are known to be equivalent to Select-Project-Join queries. The problem of evaluating such queries is NP-hard in general, but it is feasible in polynomial time on the class of acyclic queries (we omit “conjunctive,” hereafter), which was the subject of many seminal research works since the early ages of database theory (see, e.g., [11]). This class contains all queries Q whose associated query hypergraph \mathcal{H}_Q is acyclic,¹ where \mathcal{H}_Q is a hypergraph having the variables of Q as its nodes, and the (sets of variables occurring in the) atoms of Q as its hyperedges. In fact, queries arising from real applications are hardly precisely acyclic. Yet, they are often not very intricate and, in fact, tend to exhibit some limited degree of cyclicity, which suffices to retain most of the nice properties of acyclic ones. Therefore, several efforts have been spent to investigate invariants that are best suited to identify nearly-acyclic hypergraphs, leading to the definition of a number of so-called (*purely*) *structural decomposition-methods*, such as the (*generalized*) *hypertree* [34], *fractional hypertree* [43], *spread-cut* [17], and *component hypertree* [36] decompositions. These methods aim at transforming a given cyclic hypergraph into an acyclic one, by organizing its edges (or its nodes) into a polynomial number of clusters, and by suitably arranging these clusters as a tree, called decomposition tree. The original problem instance can then be evaluated over such a tree of subproblems, with a cost that is exponential in the cardinality of the largest cluster, also called *width* of the decomposition, and polynomial if this width is bounded by some constant.

Despite their different technical definitions, there is a simple mathematical framework, based on the notion of *tree projection* [37], that encompasses all the above decomposition methods, as pointed out in recent works on the subject [38, 40]. In this setting, a query Q is given together with a set \mathcal{V} of atoms, called views, which are defined over the variables in Q . The question is whether (parts of) the views can be arranged as to form a tree projection (playing the role of a decomposition tree), i.e., a novel acyclic query that still “covers” Q . By representing Q and \mathcal{V} via the hypergraphs \mathcal{H}_Q and $\mathcal{H}_\mathcal{V}$, where hyperedges one-to-one correspond with query atoms and views, respectively, the tree projection problem reveals its graph-theoretic nature. For a pair of hypergraphs $\mathcal{H}_1, \mathcal{H}_2$, let $\mathcal{H}_1 \leq \mathcal{H}_2$ denote that each hyperedge of \mathcal{H}_1 is contained in some hyperedge

(Francesco Scarcello)

¹For completeness, observe that different notions of hypergraph acyclicity have been proposed in the literature. This paper follows the standard definition of acyclic conjunctive queries, so that hypergraph acyclicity always refers to the most liberal notion, known as α -acyclicity [22].

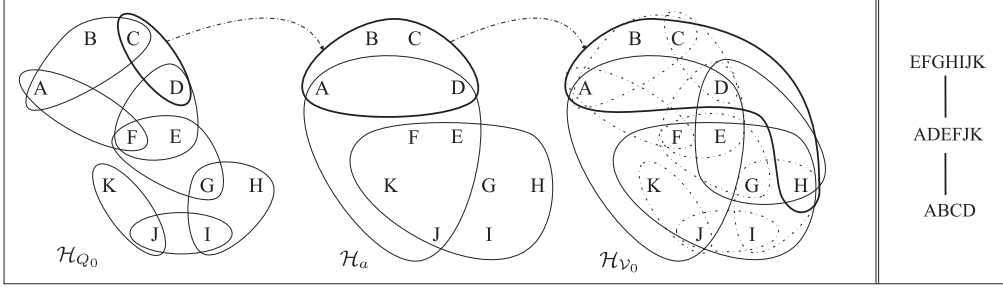


Figure 1: A tree projection \mathcal{H}_a of \mathcal{H}_{Q_0} w.r.t. \mathcal{H}_{V_0} ; On the right: A join tree JI_a for \mathcal{H}_a .

of \mathcal{H}_2 . Then, a tree projection of \mathcal{H}_Q w.r.t. \mathcal{H}_V is any acyclic hypergraph \mathcal{H}_a such that $\mathcal{H}_Q \leq \mathcal{H}_a \leq \mathcal{H}_V$. If such a hypergraph exists, then we say that the pair of hypergraphs $(\mathcal{H}_Q, \mathcal{H}_V)$ has a tree projection.²

Example 1.1. Consider the conjunctive query

$$Q_0 : r_1(A, B, C) \wedge r_2(A, F) \wedge r_3(C, D) \wedge r_4(D, E, F) \wedge \\ r_5(E, F, G) \wedge r_6(G, H, I) \wedge r_7(I, J) \wedge r_8(J, K),$$

whose associated hypergraph \mathcal{H}_{Q_0} is depicted in Figure 1, together with other hypergraphs that are discussed next.

To answer Q_0 , assume that a set \mathcal{V}_0 of views is available comprising some views, called *query views*, playing the role of query atoms, plus four additional views. The set of variables of each view is a hyperedge in the hypergraph \mathcal{H}_{V_0} (query views are depicted as dashed hyperedges). In the middle between \mathcal{H}_{Q_0} and \mathcal{H}_{V_0} , Figure 1 reports the hypergraph \mathcal{H}_a which covers \mathcal{H}_{Q_0} , and which is in its turn covered by \mathcal{H}_{V_0} —e.g., $\{C, D\} \subseteq \{A, B, C, D\} \subseteq \{A, B, C, D, H\}$. Since \mathcal{H}_a is in addition acyclic, \mathcal{H}_a is a tree projection of \mathcal{H}_{Q_0} w.r.t. \mathcal{H}_{V_0} . \triangleleft

Observe that, in the tree projection framework, views can be arbitrary, i.e., they do not depend on the specific conjunctive query Q , and can be reused to answer different queries. In particular, views may be the materialized output of any procedure over the database, possibly much more powerful than conjunctive queries. Moreover, it is known and easy to see that any decomposition method based on clustering subproblems can be viewed as an instance of this general setting, identifying a specific set of views to answer a given query Q efficiently (see Section 2).

1.2. Islands of Tractability

An *island of tractability* (cf. [47]) in the tree projection framework is a class \mathcal{C} of pairs (Q, \mathcal{V}) that can be efficiently recognized, i.e., we can check in polynomial time whether

²Note that the only known decomposition technique that does not fit the above framework is the one based on the *submodular width* [50]. This method is in fact not “purely” structural, in that the views \mathcal{V} , together with suitable associated database relations, are computed in fixed-parameter polynomial time (hence, not in polynomial-time, in general) by using the actual database over which Q has to be evaluated, rather than looking at \mathcal{H}_Q only.

a given pair actually belongs to \mathcal{C} , and such that Q can be efficiently evaluated on every database, by possibly exploiting the views that are available in \mathcal{V} .

Many specializations of tree projections, such as *tree decompositions* [53], *hypertree decompositions* [34], *component decompositions* [36], and *spread-cuts decompositions* [17], define islands of tractability whenever some fixed bound is imposed on their widths. This is also the case for *fractional hypertree decompositions* [43], whenever the resources sufficient for computing their $O(w^3)$ approximation [49] are used as available views. However, this is not the case for general tree projections. Indeed, while Goodman and Shmueli [37] observed that queries that admit a tree projection can be evaluated in polynomial time, Gottlob et al. [36] proved that checking whether a tree projection exists or not is an NP-hard problem. Hence, the class $\mathcal{C}_{tp} = \{(Q, \mathcal{V}) \mid \mathcal{H}_Q \text{ has a tree projection w.r.t. } \mathcal{H}_\mathcal{V}\}$, which includes all the above mentioned islands of tractability, is not an island of tractability in its turn. A natural question is, therefore, whether there is any subclass of \mathcal{C}_{tp} , at least including all the tractable classes mentioned above, which identifies an actual island of tractability where tree projections can be computed efficiently.

In the paper, we address the above question. The starting point of our analysis is the game-theoretic characterization of tree projections in terms of the *Robber and Captain* game [38]. The game is played on a pair of hypergraphs $(\mathcal{H}_1, \mathcal{H}_2)$ by a Captain controlling, at each move, a squads of cops encoded as the nodes in a hyperedge $h \in \text{edges}(\mathcal{H}_2)$, and by a Robber who stands on a node and can run at great speed along the edges of \mathcal{H}_1 , while being not permitted to run through a node that is controlled by a cop. In particular, the Captain may ask any cop in the squad h to run in action, as long as they occupy nodes that are currently reachable by the Robber, thereby blocking an escape path for the Robber. While cops move, the Robber may run through those positions that are left by cops or not yet occupied. The goal of the Captain is to place a cop on the node occupied by the Robber, while the Robber tries to avoid her capture. The Captain has a winning strategy if, and only if, there is a tree projection of \mathcal{H}_1 w.r.t. \mathcal{H}_2 .

Based on the above characterization, we proceed as follows:

- We define the notion of *greedy strategies*, which are winning strategies for the Captain, possibly non-monotone, where it is required that *all* cops available at the current squad h and reachable by the Robber enter in action. If all of them are in action, then a new squad h' is selected, again requiring that all the active cops, i.e., those in the frontier, enter in action. In the Robber and Captain game, it is known that there is no incentive for the Captain to play a strategy that is not monotone [38]. Instead, by focusing on greedy strategies, we can exhibit examples where there exists non-monotone winning strategies but no monotone winning one.
- We show that *greedy strategies* can be computed in polynomial time, and that based on them (even on non-monotone ones) it is possible to construct, again in polynomial time, tree projections, which are called *greedy*. Therefore, the class $\mathcal{C}_{gtp} \subset \mathcal{C}_{tp}$ of all greedy tree projections turns out to be an island of tractability.
- We show that \mathcal{C}_{gtp} properly includes most previously known islands of tractability (based on structural properties), precisely because of the power of non-monotonic strategies. Indeed, (arbitrary) non-monotone strategies do not correspond in general to valid decompositions in the games characterizing such islands of tractability, which are in fact defined in terms of monotone strategies only. The novel notion

of greedy tree projections allows us to define new islands of tractability from any known structural decomposition method. In particular, from the notion of generalized hypertree decomposition, we obtain the novel notion of *greedy (generalized) hypertree decomposition*, that is tractable and strictly more powerful than the hypertree decomposition (which is instead characterized by a monotonic hypergraph game).

- Finally, by using the game theoretic characterization of tree projections, we pinpoint that dealing with this general NP-hard notion is fixed-parameter tractable if the maximum arity of views is used as the parameter. Even this result can be useful in real-world applications, since the case of small arity structures is quite frequent in practice.

Organization. The rest of paper is organized as follows. Section 2 illustrates some basic notions and concepts. Greedy strategies for the Robber and Captain game are introduced and analyzed in Section 3, and based on them islands of tractability for tree projections are singled out in Section 4. Specializations of the results to known structural decomposition methods (as well as to structures having “small” arities) are discussed in Section 5. Literature related to “Cops and Robbers” games is illustrated in Section 7, while a few remarks and open issues are discussed in Section 8.

2. Preliminaries

Hypergraphs and Acyclicity. A *hypergraph* \mathcal{H} is a pair (V, H) , where V is a finite set of nodes and H is a set of hyperedges such that, for each $h \in H$, $h \subseteq V$. If $|h| = 2$ for each (hyper)edge $h \in H$, then \mathcal{H} is a *graph*. We assume without loss of generality that every node occurs in some hyperedge, that is, $V = \bigcup_{h \in H} h$. We denote V and H by *nodes*(\mathcal{H}) and *edges*(\mathcal{H}), respectively.

A hypergraph \mathcal{H} is *acyclic* (more precisely, α -acyclic [22]) if, and only if, it has a join tree [12]. A *join tree* \mathcal{JT} for a hypergraph \mathcal{H} is a tree whose vertices are the hyperedges of \mathcal{H} such that, whenever a node $X \in V$ occurs in two hyperedges h_1 and h_2 of \mathcal{H} , then h_1 and h_2 are connected in \mathcal{JT} , and X occurs in each vertex on the unique path linking h_1 and h_2 . In words, the set of vertices in which X occurs induces a (connected) subtree of \mathcal{JT} . We will refer to this condition as the *connectedness condition* of join trees.

Example 2.1. Consider the hypergraph \mathcal{H}_a reported in Figure 1. We have $\text{nodes}(\mathcal{H}_a) = \{A, B, C, D, E, F, G, H, I, J, K\}$ and $\text{edges}(\mathcal{H}_a) = \{\{A, B, C, D\}, \{A, D, E, F, J, K\}, \{E, F, G, H, I, J, K\}\}$. The hypergraph is acyclic, as it is witnessed by the join tree \mathcal{JT}_a depicted on the right part of the same figure. \triangleleft

Tree Decompositions of (Hyper)graphs. Several efforts have been spent in the literature to investigate hypergraph properties that are best suited to identify nearly-acyclic hypergraphs, leading to the definition of a number of so-called (*purely*) *structural decomposition methods*. Within these methods, the notions of tree decomposition and treewidth [53] represent a significant success story in Computer Science (see, e.g., [31]), which are meant to provide a measure of the degree of cyclicity in *graphs*.

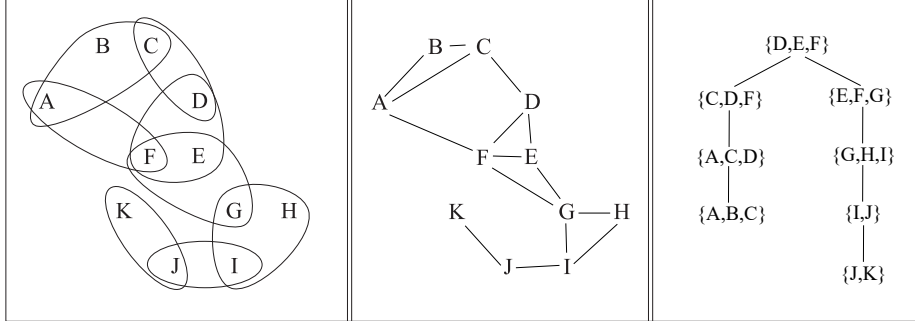


Figure 2: The hypergraph \mathcal{H}_{Q_0} , its Gaifman graph, and a tree decomposition of it.

A *tree decomposition* [53] of a graph G is a pair $\langle T, \chi \rangle$, where $T = (N, E)$ is a tree, and χ is a labeling function assigning to each vertex $v \in N$ a set of vertices $\chi(v) \subseteq \text{nodes}(G)$, such that the following conditions are satisfied: (1) for each node $Y \in \text{nodes}(G)$, there exists $p \in N$ such that $Y \in \chi(p)$; (2) for each edge $\{X, Y\} \in \text{edges}(G)$, there exists $p \in N$ such that $\{X, Y\} \subseteq \chi(p)$; and (3) for each node $Y \in \text{nodes}(G)$, the set $\{p \in N \mid Y \in \chi(p)\}$ induces a (connected) subtree of T . The *width* of $\langle T, \chi \rangle$ is the number $\max_{p \in N} (|\chi(p)| - 1)$.

For the application of the notion of treewidth over an arbitrary hypergraph, it is necessary to deal with a graph-based representation of its associated hypergraph. There are a number of possible choices, and we next focus on the simplest and widely used one.

The *Gaifman graph* of a hypergraph \mathcal{H} is defined over the set $\text{nodes}(\mathcal{H})$ of the nodes of \mathcal{H} , and contains an edge $\{X, Y\}$ if, and only if, $\{X, Y\} \subseteq h$ holds, for some hyperedge $h \in \text{edges}(\mathcal{H})$. The *treewidth* of \mathcal{H} is the minimum width over all the tree decompositions of its Gaifman graph. Deciding whether a given hypergraph has treewidth bounded by a fixed natural number k is known to be feasible in linear time [13].

Example 2.2. Consider the hypergraph \mathcal{H}_{Q_0} discussed in Example 1.1 and reported again in Figure 2, for the sake of readability. The hypergraph \mathcal{H}_{Q_0} is not acyclic, as it is not possible to build a join tree for it. In fact, Figure 2 also reports the Gaifman graph of \mathcal{H}_{Q_0} and a tree decomposition of it. Note that there are vertices of the tree decomposition containing 4 nodes of \mathcal{H}_{Q_0} . Indeed, the treewidth of \mathcal{H}_{Q_0} is 3. \triangleleft

(Generalized) Hypertree Decompositions of Hypergraphs. A crucial limitation for the practical use of the tree decomposition method is that it applies to graph representations only, hence obscuring in many cases the actual degree of cyclicity of the original hypergraph. For instance, for the acyclic hypergraph \mathcal{H}_a depicted in Figure 1, the Gaifman graph contains a clique over the variables in $\{A, D, E, F, J, K\}$, since all of them occur together in one hyperedge. Hence, the treewidth of this acyclic hypergraph is 5. Motivated by this observation, specific width-notions for hypergraphs have been defined and studied, and often these are more effective than simply applying the treewidth on a suitable “binarization” [39]. In particular, the natural counterpart of the tree decomposition method over hypergraphs is the notion of (generalized) hypertree decomposition [35] (see [32] for a survey on recent advances and applications).

A *hypertree for a hypergraph* \mathcal{H} is a triple $\langle T, \chi, \lambda \rangle$, where $T = (N, E)$ is a rooted tree, and χ and λ are labeling functions which associate each vertex $p \in N$ with two

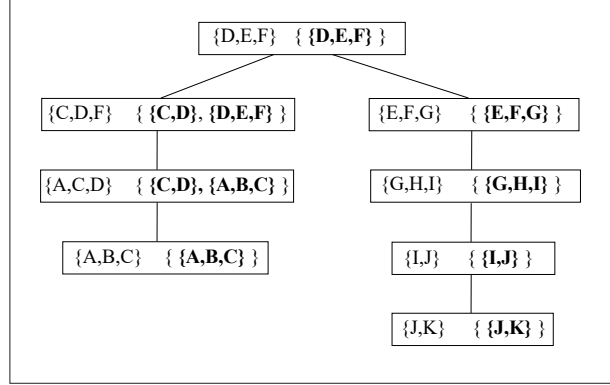


Figure 3: A (generalized) hypertree decomposition of the hypergraph \mathcal{H}_{Q_0} .

sets $\chi(p) \subseteq \text{nodes}(\mathcal{H})$ and $\lambda(p) \subseteq \text{edges}(\mathcal{H})$. If $T' = (N', E')$ is a subtree of T , we define $\chi(T') = \bigcup_{v \in N'} \chi(v)$. In the following, for any rooted tree T , we denote the set of vertices N of T by $\text{vertices}(T)$, and the root of T by $\text{root}(T)$. Moreover, for any $p \in N$, T_p denotes the subtree of T rooted at p .

A *generalized hypertree decomposition* [35] of a hypergraph \mathcal{H} is a hypertree $HD = \langle T, \chi, \lambda \rangle$ for \mathcal{H} such that: (1) for each hyperedge $h \in \text{edges}(\mathcal{H})$, there exists $p \in \text{vertices}(T)$ such that $h \subseteq \chi(p)$; (2) for each node $Y \in \text{nodes}(\mathcal{H})$, the set $\{p \in \text{vertices}(T) \mid Y \in \chi(p)\}$ induces a (connected) subtree of T ; and (3) for each $p \in \text{vertices}(T)$, $\chi(p) \subseteq \text{nodes}(\lambda(p))$. The *width* of a generalized hypertree decomposition $\langle T, \chi, \lambda \rangle$ is $\max_{p \in \text{vertices}(T)} |\lambda(p)|$. The *generalized hypertree width* $ghw(\mathcal{H})$ of \mathcal{H} is the minimum width over all its generalized hypertree decompositions. The notion is a true generalization of acyclicity, as the acyclic hypergraphs are precisely those hypergraphs having generalized hypertree width one.

Note that conditions (1) and (2) above state that $\langle T, \chi \rangle$ is a tree decomposition of the Gaifman graph of \mathcal{H} , while condition (3) prescribes that, at each vertex p , all nodes in the χ labeling are covered by hyperedges in the λ labeling. Indeed, the width of the generalized hypertree decomposition is defined in terms of the number of hyperedges used to cover the nodes, rather than of the number of such nodes, as in the width of $\langle T, \chi \rangle$.

Example 2.3. Consider again the hypergraph \mathcal{H}_{Q_0} and the tree decomposition reported in Figure 2. The nodes occurring at each vertex of the decomposition can be covered by using two hyperedges at most, as illustrated in the generalized hypertree decomposition depicted in Figure 3. Therefore, the width of this decomposition is 2 and thus $ghw(\mathcal{H}_{Q_0}) \leq 2$. Actually, $ghw(\mathcal{H}_{Q_0}) = 2$ because \mathcal{H}_{Q_0} is a cyclic hypergraph, which entails $ghw(\mathcal{H}_{Q_0}) > 1$. \triangleleft

A *hypertree decomposition* [34] of \mathcal{H} is a generalized hypertree decomposition $HD = \langle T, \chi, \lambda \rangle$ where: (4) for each $p \in \text{vertices}(T)$, $\text{nodes}(\lambda(p)) \cap \chi(T_p) \subseteq \chi(p)$. Note that the inclusion in the above condition is actually an equality, because Condition (3) implies the reverse inclusion. The *hypertree width* $hw(\mathcal{H})$ of \mathcal{H} is the minimum width over all its hypertree decompositions. Let k be any fixed natural number. For any hypergraph \mathcal{H} , deciding whether $hw(\mathcal{H}) \leq k$ is feasible in polynomial time (and, actually, it is highly-parallelizable) [34], while deciding whether $ghw(\mathcal{H}) \leq k$ is NP-complete [36].

Therefore, condition (4) plays the technical role of guaranteeing that the hypertree decomposition is a tractable structural method. Moreover, it cannot be much larger than its generalized variant, since $ghw(\mathcal{H}) \leq hw(\mathcal{H}) \leq 3 \times ghw(\mathcal{H}) + 1$ holds [5]. As an example, the reader might check that $hw(\mathcal{H}_{Q_0}) = 2$, too, because the generalized hypertree decomposition depicted in Figure 3 satisfies condition (4), and thus it is actually a hypertree decomposition. Later in the paper, Figure 9 shows a hypergraph where the generalized hypertree width is strictly smaller than the hypertree width.

Tree Projections. The framework of the tree projections is a mathematical framework that encompasses all (purely) structural decomposition methods defined in the literature. Formally, for two hypergraphs \mathcal{H}_1 and \mathcal{H}_2 , we write $\mathcal{H}_1 \leq \mathcal{H}_2$ if, and only if, each hyperedge of \mathcal{H}_1 is contained in at least one hyperedge of \mathcal{H}_2 . Let $\mathcal{H}_1 \leq \mathcal{H}_2$; then, a *tree projection* of \mathcal{H}_1 with respect to \mathcal{H}_2 is an acyclic hypergraph \mathcal{H}_a such that $\mathcal{H}_1 \leq \mathcal{H}_a \leq \mathcal{H}_2$. If such a hypergraph \mathcal{H}_a exists, then we say that the pair $(\mathcal{H}_1, \mathcal{H}_2)$ has a tree projection. See Figure 1 for an example. Without loss of generality, we assume hereafter that \mathcal{H}_1 and \mathcal{H}_2 have the same set of nodes. Indeed, $nodes(\mathcal{H}_1) \not\subseteq nodes(\mathcal{H}_2)$ trivially entails that there are no tree projections of \mathcal{H}_1 with respect to \mathcal{H}_2 , while $nodes(\mathcal{H}_2) \not\subseteq nodes(\mathcal{H}_1)$ entails that there are useless nodes in \mathcal{H}_2 .

According to this unifying view, differences among the various (purely) structural decomposition methods just come in the way the resource hypergraph \mathcal{H}_2 is defined. For instance, given a hypergraph \mathcal{H} and a natural number $k > 0$, let \mathcal{H}^k denote the hypergraph over the same set of nodes as \mathcal{H} , and whose set of hyperedges is given by all possible unions of k edges in \mathcal{H} , i.e., $edges(\mathcal{H}^k) = \{h_1 \cup h_2 \cup \dots \cup h_k \mid \{h_1, h_2, \dots, h_k\} \subseteq edges(\mathcal{H})\}$. Then, it is well known and easy to see that $ghw(\mathcal{H}) \leq k$ if, and only if, there is a tree projection for $(\mathcal{H}, \mathcal{H}^k)$.

Similarly, let \mathcal{H}^{tk} be the hypergraph over the same set of nodes as \mathcal{H} , and whose set of hyperedges is given by all possible clusters $B \subseteq nodes(\mathcal{H})$ of nodes such that $|B| \leq k + 1$. Then, \mathcal{H} has treewidth at most k if, and only if, there is a tree projection for $(\mathcal{H}, \mathcal{H}^{tk})$.

However, the notion of tree projection is more general than both treewidth and hypertree width, because the hyperedges of the resource hypergraph may model arbitrary subproblems of the given instance whose solutions are easy to compute, or already available from previous computations. For instance, in Example 1.1, the resource hypergraph $\mathcal{H}_{\mathcal{V}_0}$ does not correspond to any of the above mentioned decomposition methods.

Conjunctive Queries. We leave the section by recalling conjunctive queries and their hypergraph-based representation, over which structural decomposition methods can be applied—as introduced in Example 1.1.

A *conjunctive query* Q consists of a finite conjunction of atoms of the form $r_1(\mathbf{u}_1) \wedge \dots \wedge r_m(\mathbf{u}_m)$, where r_1, \dots, r_m (with $m > 0$) are relation symbols (not necessarily distinct), and $\mathbf{u}_1, \dots, \mathbf{u}_m$ are lists of terms (i.e., variables or constants). The set of all atoms in Q is denoted by $atoms(Q)$. For a set of atoms A , $vars(A)$ is the set of variables occurring in the atoms in A . For short, $vars(Q)$ denotes $vars(atoms(Q))$.

There is a very natural way to associate a hypergraph $\mathcal{H}_{\mathcal{V}} = (N, H)$ with any set \mathcal{V} of atoms: the set N of nodes consists of all variables occurring in \mathcal{V} ; for each atom in \mathcal{V} , the set H of hyperedges contains a hyperedge including all its variables; and no other hyperedge is in H . For a query Q , the hypergraph associated with $atoms(Q)$ is briefly denoted by \mathcal{H}_Q . If \mathcal{H}_Q is a connected hypergraph, we say that Q is a *connected* query.

3. Greedy Strategies in Robber and Captain Games

In this section, we define the concept of greedy strategies in the game-theoretic characterization of tree projections proposed in [38], and we show that, unlike arbitrary strategies, greedy ones can be efficiently computed.

To formalize our results, we need to introduce some additional definitions and notations, which will be intensively used in the following.

Assume that a hypergraph \mathcal{H} is given. Let V , W , and $\{X, Y\}$ be sets of nodes. Then, X is said $[V]$ -adjacent (in \mathcal{H}) to Y if there exists a hyperedge $h \in \text{edges}(\mathcal{H})$ such that $\{X, Y\} \subseteq (h - V)$. A $[V]$ -path from X to Y is a sequence $X = X_0, \dots, X_\ell = Y$ of nodes such that X_i is $[V]$ -adjacent to X_{i+1} , for each $i \in [0 \dots \ell-1]$. We say that X $[V]$ -touches Y if X is $[\emptyset]$ -adjacent to $Z \in \text{nodes}(\mathcal{H})$, and there is a $[V]$ -path from Z to Y ; similarly, X $[V]$ -touches the set W if X $[V]$ -touches some node $Y \in W$. We say that W is $[V]$ -connected if $\forall X, Y \in W$ there is a $[V]$ -path from X to Y . A $[V]$ -component (of \mathcal{H}) is a maximal $[V]$ -connected non-empty set of nodes $W \subseteq (\text{nodes}(\mathcal{H}) - V)$. For any $[V]$ -component C , let $\text{edges}(C) = \{h \in \text{edges}(\mathcal{H}) \mid h \cap C \neq \emptyset\}$, and for a set of hyperedges $H \subseteq \text{edges}(\mathcal{H})$, let $\text{nodes}(H)$ denote the set of nodes occurring in H , that is $\text{nodes}(H) = \bigcup_{h \in H} h$. For any component C of \mathcal{H} , we denote by $\text{Fr}(C, \mathcal{H})$ the *frontier* of C (in \mathcal{H}), i.e., the set $\text{nodes}(\text{edges}(C))$.³ Moreover, $\partial(C, \mathcal{H})$ denote the *border* of C (in \mathcal{H}), i.e., the set $\text{Fr}(C, \mathcal{H}) \setminus C$. Note that $C_1 \subseteq C_2$ entails $\text{Fr}(C_1, \mathcal{H}) \subseteq \text{Fr}(C_2, \mathcal{H})$.

In the following sections, given any pair of hypergraphs $(\mathcal{H}_1, \mathcal{H}_2)$ and a set of nodes $C \subseteq \mathcal{H}_1$, we write for short $\text{Fr}(C)$ and ∂C to denote $\text{Fr}(C, \mathcal{H}_1)$ and $\partial(C, \mathcal{H}_1)$, respectively.

3.1. Game-Theoretic Characterization

The *Robber and Captain* game is played on a pair of hypergraphs $(\mathcal{H}_1, \mathcal{H}_2)$ by a Robber and a Captain controlling some squads of cops, in charge of the surveillance of a number of strategic targets. The Robber stands on a node and can run at great speed along the edges of \mathcal{H}_1 . However, (s)he is not permitted to run through a node that is controlled by a cop. Each move of the Captain involves one squad of cops, which is encoded as a hyperedge $h \in \text{edges}(\mathcal{H}_2)$. The Captain may ask some cops in the squad h to run in action, as long as they occupy nodes that are currently reachable by the Robber, thereby blocking an escape path for the Robber. Thus, “second-lines” cops cannot be activated by the Captain. Note that the Robber is fast and may see cops that are entering in action. Therefore, while cops move, the Robber may run through those positions that are left by cops or not yet occupied. The goal of the Captain is to place a cop on the node occupied by the Robber, while the Robber tries to avoid her/his capture.

Definition 3.1. Let \mathcal{H}_1 and \mathcal{H}_2 be two hypergraphs. The Robber and Captain game on $(\mathcal{H}_1, \mathcal{H}_2)$ is formalized as follows. A *position* for the Captain is a pair (h, M) where h is a hyperedge of \mathcal{H}_2 and $M \subseteq h$. A *configuration* is a triple (h, M, C) , where (h, M) is a position for the Captain, and C is the $[M]$ -component where the Robber stands.⁴ The initial configuration is $(\emptyset, \emptyset, \text{nodes}(\mathcal{H}_1))$.

³The choice of the term “frontier” to name the union of a component with its outer border is due to the role that this notion plays in the hypergraph game described in the subsequent section.

⁴It is easy to see that in such games, being the robber arbitrarily fast, what matters is not the precise node where the robber stands, but just the $[M]$ -component where (s)he is free to move.

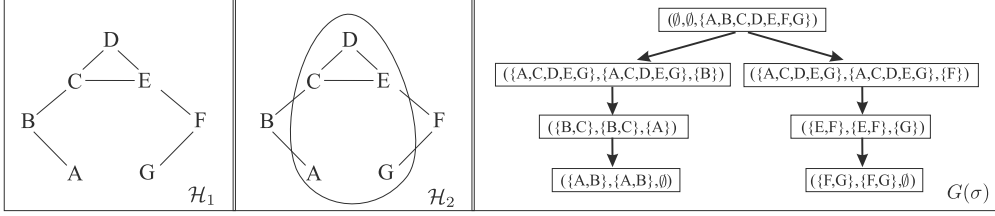


Figure 4: The hypergraphs \mathcal{H}_1 and \mathcal{H}_2 , plus the graph $G(\sigma)$ in Example 3.2.

A *strategy* σ is a function that encodes the *moves* of the Captain. Its domain includes the initial configuration. For each configuration $v_p = (h_p, M_p, C_p)$ in the domain of σ , $\sigma(v_p) = (h_r, M_r)$, with $M_r \subseteq h_r \cap \text{Fr}(C_p)$, is the novel position for the Captain. After this move, the Robber can select any $[v_p, M_r]$ -option, i.e., any $[M_r]$ -component C_r such that $C_p \cup C_r$ is $[M_p \cap M_r]$ -connected. If there is no $[v_p, M_r]$ -option, then (h_r, M_r, \emptyset) is said a *capture* configuration induced by σ . The move of the Captain is *monotone* if, for each $[v_p, M_r]$ -option C_r , $C_r \subseteq C_p$. The domain of σ includes the configuration (h_r, M_r, C_r) , for each $[v_p, M_r]$ -option C_r . No other configuration is in the domain of σ . The strategy σ is monotone if it encodes only monotone moves over the configurations in its domain.

A strategy σ can be represented as a directed graph $G(\sigma) = (N, A)$, called *strategy graph*, as follows. The set N of nodes is the set of all configurations in the domain of σ plus all capture configurations induced by σ . If $v_p = (h_p, M_p, C_p)$ is a configuration and $\sigma(v_p) = (h_r, M_r)$, then A contains an arc from v_p to (h_r, M_r, C_r) for each $[v_p, M_r]$ -option C_r , and to (h_r, M_r, \emptyset) if there is no $[v_p, M_r]$ -option. We say that σ is a winning strategy (for the Captain) if $G(\sigma)$ is acyclic. Otherwise, i.e., if $G(\sigma)$ contains a cycle, then the Robber can avoid her/his capture forever.

Example 3.2. Consider the two hypergraphs \mathcal{H}_1 and \mathcal{H}_2 reported in Figure 4, together with the strategy graph $G(\sigma)$. The graph encodes a winning strategy σ for the Captain. From the initial configuration $(\emptyset, \emptyset, \text{nodes}(\mathcal{H}_1))$, the Captain activates all the cops in the hyperedge $\{A, C, D, E, G\}$, so that the Robber has two available options, i.e., $\{B\}$ and $\{F\}$. In the former (resp., latter) case, the Captain activates all the cops in the hyperedge $\{B, C\}$ (resp., $\{E, F\}$), so that the Robber has necessarily to occupy the node A (resp., G). Finally, the Captain activates the cops in $\{A, B\}$ (resp., $\{F, G\}$) and captures the Robber. Note that the strategy σ is non-monotone, because the Robber is allowed to return on A and G , after that these nodes have been previously occupied by the Captain in the first move. \triangleleft

In the above example, the hyperedge $\{A, C, D, E, G\}$ of \mathcal{H}_2 “absorbs” the cycle in \mathcal{H}_1 , so that it is easily seen that there is a tree projection \mathcal{H}_a of \mathcal{H}_1 w.r.t. \mathcal{H}_2 (see Figure 5). The fact that on this pair the Captain has a winning strategy is not by chance.

Theorem 3.3 ([38]). *There is a tree projection of \mathcal{H}_1 w.r.t. \mathcal{H}_2 if, and only if, there is a winning strategy in the Robber and Captain game played on $(\mathcal{H}_1, \mathcal{H}_2)$.*

Recall that the winning strategy in Example 3.2 is not monotone. However, an important property of this game is that there is no incentive for the Captain to play a strategy that is not monotone.

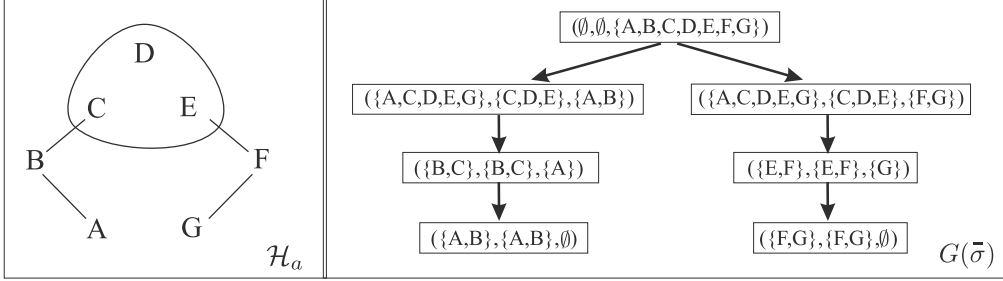


Figure 5: A tree projection \mathcal{H}_a for the pair in Example 3.2, plus the graph $G(\bar{\sigma})$.

Theorem 3.4 (cf. [38]). *In the Robber and Captain game played on the pair $(\mathcal{H}_1, \mathcal{H}_2)$, a winning strategy exists if, and only if, a monotone winning strategy exists.*

Moreover, from any monotone winning strategy, a tree projection of \mathcal{H}_1 w.r.t. \mathcal{H}_2 can be computed in polynomial time.

Example 3.5. Consider again the setting of Example 3.2, and the strategy graph $G(\bar{\sigma})$ shown in Figure 5. Note that the strategy $\bar{\sigma}$ is monotone, and in fact the moves of the Captain one-to-one correspond to the hyperedges in the tree projection \mathcal{H}_a . \triangleleft

The crucial properties to establish Theorem 3.4 are next recalled, as they will be useful in our subsequent analysis too. Let σ be a strategy, and let $v_p = (h_p, M_p, C_p)$ and $v_r = (h_r, M_r, C_r)$ be two configurations in its domain such that $\sigma(v_p) = (h_r, M_r)$ and C_r is a $[v_p, M_r]$ -option. Let $\sigma(v_r) = (h_s, M_s)$ and define $\text{ED}((M_r, C_r), M_s) = M_r \cap \text{Fr}(C_r) \setminus M_s$ (which is equivalent to $\partial C_r \setminus M_s$ because C_r is an $[M_r]$ -component) as the *escape-door* of the Robber in v_r when attacked with M_s . From [38], a move is monotone if, and only if, such an escape door is empty; in particular, $\sigma(v_r)$ is non-monotone if (and only if) $\text{ED}((M_r, C_r), M_s) \neq \emptyset$. Let $M'_r = M_r \setminus \text{ED}((M_r, C_r), M_s)$, let C'_r be the $[M'_r]$ -component with $C_r \cup \text{ED}((M_r, C_r), M_s) \subseteq C'_r$, which exists since $\text{ED}((M_r, C_r), M_s) \subseteq \text{Fr}(C_r)$ and $M'_r \subseteq M_r$, and let $v'_r = (h_r, M'_r, C'_r)$. Finally, consider the following strategy σ' :

$$\sigma'(h, M, C) = \begin{cases} (h_r, M'_r) & \text{if } (h, M, C) = (h_p, M_p, C_p) \\ \sigma(h, M, C) & \text{otherwise.} \end{cases} \quad (1)$$

For such a state of the game, a number of technical properties have been proved in [38]. We summarize them in the following lemma.

Lemma 3.6 ([38]). *The following properties hold:*

- (1) $\text{ED}((M'_r, C'_r), M_s) = \emptyset$.
- (2) For each $[v_p, M_r]$ -option C , either $C \subseteq C'_r$ or C is a $[v_p, M'_r]$ -option.
- (3) For each $[v_p, M'_r]$ -option $C' \neq C'_r$, C' is a $[v_p, M_r]$ -option.
- (4) A set C is a $[v_r, M_s]$ -option if, and only if, it is a $[v'_r, M_s]$ -option.
- (5) If σ is a winning strategy, then σ' is a winning strategy too.

3.2. Greedy Strategies

Since winning strategies correspond to tree projections, there is no efficient algorithm for their computation. Indeed, just recall that deciding the existence of a tree projection is not feasible in polynomial time, unless $P = NP$ [36].

Our goal is then to focus on certain “greedy” strategies that are easy to compute. Intuitively, in greedy strategies it is required that *all* cops available at the current squad h_p and reachable by the Robber enter in action. If all of them are in action, then a new squad h_r is selected, again requiring that all the active cops, i.e., those in the frontier, enter in action.

Definition 3.7. On the Robber and Captain game played on $(\mathcal{H}_1, \mathcal{H}_2)$, a strategy σ is *greedy* if, for any configuration $v_p = (h_p, M_p, C_p)$ in the domain of σ , the next position $\sigma(v_p) = (h_r, M_r)$ is such that $M_r = h_r \cap \text{Fr}(C_p)$, where $h_r = h_p$ if $h_p \cap C_p \neq \emptyset$, and h_r is any squad in $\text{edges}(\mathcal{H}_2)$ if $h_p \cap C_p = \emptyset$.

Given such a greedy way to select cops at each step, observe that the former case ($h_p \cap C_p \neq \emptyset$) may only occur if the Robber is able to come back to some position previously controlled by the Captain. Greedy winning strategies are indeed non-monotone in general, and for some pair of hypergraphs it is possible that there is no monotone winning greedy strategy, although monotone winning strategies (non-greedy) exist.

Example 3.8. Consider again the hypergraphs \mathcal{H}_1 and \mathcal{H}_2 shown in Figure 4, and recall that the strategy graph of a monotone winning strategy $\bar{\sigma}$ is depicted in Figure 5. However, there is no monotone greedy strategy in this case. Indeed, if at the beginning of the game the Captain asks the squad $\{A, C, D, E, G\}$ to enter in action and the Robber goes on B , then in the next move the Robber is forced to lose the control on A in order to move on $\{C, B\}$ and eventually win via $\{B, A\}$ —see again Figure 4. On the other hand, if the attack of the Captain starts on either side, say on the left branch, the Captain has then to attack the component that includes the triangle and the other branch. At this point, the only available greedy choice is use the big squad and hence to employ cops $\{C, D, E, G\}$. However, as in the previous case, G will be later (necessarily) left free to the Robber, in order to win the game. \triangleleft

We now show that, differently from arbitrary strategies, the existence of greedy winning strategies can be decided in polynomial time. To establish the result, a useful technical property is that greedy strategies can only involve a polynomial number of configurations. Let us denote by $\text{MaxGreedyStrat}(\mathcal{H}_1, \mathcal{H}_2)$ the maximum domain cardinality over any greedy strategy in the Robber and Captain game on a pair $(\mathcal{H}_1, \mathcal{H}_2)$.

Lemma 3.9. *Let $(\mathcal{H}_1, \mathcal{H}_2)$ be a pair of hypergraphs. Then, $\text{MaxGreedyStrat}(\mathcal{H}_1, \mathcal{H}_2)$ is at most $|\text{edges}(\mathcal{H}_2)| \times |\text{nodes}(\mathcal{H}_1)|(|\text{edges}(\mathcal{H}_2)| \times |\text{nodes}(\mathcal{H}_1)| + 1) + 1$.*

Proof. Let σ be a greedy strategy, and let $v_p = (h_p, M_p, C_p)$ be a configuration in its domain. Note that the only configuration where $h_p = M_p = \emptyset$ is the starting configuration $(\emptyset, \emptyset, \text{nodes}(\mathcal{H}_1))$, which is taken into account by the final “+1” in the statement. Therefore, we next assume $M_p \neq \emptyset$. In order to establish the result, we shall derive an upper bound on the number of possible distinct configurations (h_r, M_r, C_r) following (h_p, M_p, C_p) in the game where the Captain plays according to σ . In particular, we shall

distinguish two cases, based on whether $h_p \cap C_p$ is empty or not. For each of these two scenarios, we shall bound the number of such configurations (h_r, M_r, C_r) according to the constraints imposed on them by the definition of greedy strategy (cf. Definition 3.7).

Consider the case where $h_p \cap C_p = \emptyset$. In this case, a new squad $h_r \in \text{edges}(\mathcal{H}_2)$ is chosen by the Captain according to σ . Since C_p is an $[M_p]$ -component and thus $\partial C_p \subseteq M_p \subseteq h_p$, we get that this case occurs only if C_p is actually an $[h_p]$ -component, too. Such a component is uniquely identified by any pair of the form (h_p, X_p) such that $X_p \in \text{nodes}(\mathcal{H}_1)$ is a representative of the component (e.g., the node in C_p having the smallest position according to any fixed ordering over the nodes). It follows that the new set of cops $M_r = h_r \cap \text{Fr}(C_i)$ is uniquely determined by h_r and C_p and thus may be identified through a triple (h_r, h_p, X_p) . Thus, the maximum number of such sets M_r of cops is $|\text{edges}(\mathcal{H}_2)|^2 \times |\text{nodes}(\mathcal{H}_1)|$. Moreover, the possible configurations (h_r, M_r, C_r) following (h_p, M_p, C_p) in the game where the Captain plays according to σ are identified by quadruples of the form (h_r, h_p, X_p, X_r) , where h_r is used both to identify itself and to determine the set M_r together with h_p and X_p , and where X_r is a representative of the $[M_r]$ -component. In fact, if there is no $[v_p, M_p]$ -option, then X_r is a distinguished element not in $\text{nodes}(\mathcal{H}_1)$ (or some element in M_p occupied by some cop) meaning that the only configuration following (h_p, M_p, C_p) is (h_r, M_r, \emptyset) where the Robber is captured. Overall, the maximum number of such configurations is $|\text{edges}(\mathcal{H}_2)|^2 \times |\text{nodes}(\mathcal{H}_1)|^2$.

Finally, consider the case where $h_p \cap C_p \neq \emptyset$. In this case, $M_r = h_p \cap \text{Fr}(C_p)$. Since C_p is an $[M_p]$ -component, $\partial C_p \subseteq M_p \subseteq h_p$. It follows that the new nodes from $\text{Fr}(C_p)$ to be included in M_r belong to C_p , that is, we may also write $M_r = M_p \cup (h_p \cap C_p)$. Note that no configuration of the game following this one can be of this type. Indeed, every $[M_r]$ -component C_r where the Robber may go from C_p will be a subset of C_p (because $\partial C_p \subseteq M_p \subseteq M_r \subseteq h_p$), and will have intersections with h_p . As a further consequence, such a C_r must be an $[h_p]$ -component. By contradiction, if there is some node $X_p \in C_r \subseteq C_p$ that is $[M_r]$ -connected to some X_r in $h_p \setminus M_r$, then X_p is also $[M_p]$ -connected to X_r . However, this is impossible because X_p is also in C_p and hence X_r would be in C_p , too, and hence in $h_p \cap C_p$ and in M_r , by construction. Therefore, the possible configurations (h_p, M_r, C_r) following (h_p, M_p, C_p) in the game where the Captain plays according to σ are identified by pairs of the form (h_p, X_p) , where $X_p \in \text{nodes}(\mathcal{H}_1)$ is the representative of the $[h_p]$ -component C_r (and where M_r is computed from them). As above, if there is no $[v_p, M_p]$ -option, then X_p is a distinguished element witnessing that the configuration is a capture configuration of the form (h_p, M_r, \emptyset) . Overall, the maximum number of such configurations is $|\text{edges}(\mathcal{H}_2)| \times |\text{nodes}(\mathcal{H}_1)|$. \square

To see that the existence of a winning greedy strategy is decidable in polynomial time, consider the GREEDYWINNINGSTRATEGY algorithm illustrated in Figure 6, which receives as input a configuration (h_p, M_p, C_p) for the Robber and Captain game, plus a “level” i . Note that this algorithm is a high-level specification of an alternating Turing machine, say \mathcal{M}_G [44].

After the first step, where we check that the number of recursive calls has not exceeded the number of all distinct configurations, the algorithm suddenly evidences its *non-deterministic* nature. Indeed, it guesses a hyperedge h_r corresponding to the next move of the Captain (existential step of \mathcal{M}_G). Eventually, it returns TRUE if, and only if, the recursive calls GREEDYWINNINGSTRATEGY($h_r, M_r, C_r, i+1$) with $M_r = h_r \cap \text{Fr}(C_p)$ succeed on each $[(h_p, M_p, C_p), M_r]$ -option (universal step of \mathcal{M}_G).

```

Boolean function GREEDYWINNINGSTRATEGY( $h_p, M_p, C_p, i$ );
/* ( $h_p, M_p, C_p$ ) is an extended configuration over  $(\mathcal{H}_1, \mathcal{H}_2)$ ,
 $i \geq 0$  is a natural number */

1) if  $i > \text{MaxGreedyStrat}(\mathcal{H}_1, \mathcal{H}_2)$ , then return FALSE;
2) if  $h_p \cap C_p \neq \emptyset$ , then let  $h_r = h_p$ ;
   else guess a hyperedge  $h_r \in \text{edges}(\mathcal{H}_2)$ ;
3) let  $M_r = h_r \cap \text{Fr}(C_p)$ ;
4) for each  $[(h_p, M_p, C_p), M_r]$ -option  $C_r$  do
   if not GREEDYWINNINGSTRATEGY( $h_r, M_r, C_r, i + 1$ ), then return FALSE;
5) return TRUE;

```

Figure 6: GREEDYWINNINGSTRATEGY.

Theorem 3.10. *Deciding the existence of a greedy winning strategy in the Robber and Captain game is feasible in polynomial time.*

Proof. Let $(\mathcal{H}_1, \mathcal{H}_2)$ be a pair of hypergraphs, and consider the execution of the Boolean function GREEDYWINNINGSTRATEGY on input the starting $(\emptyset, \emptyset, \text{nodes}(\mathcal{H}_1), 0)$. In general, the function receives as its input a quadruple (h_p, M_p, C_p, i) , where (h_p, M_p, C_p) is a configuration for a greedy strategy and where i counts the number of recursive invocations, i.e., the recursion level of the current invocation. For the moment, let us get rid of step (1). Then, each invocation of the algorithm, based on the current configuration (h_p, M_p, C_p) , computes the next position (h_r, M_r) of the Captain. In particular, by Definition 3.7, the position is completely determined by the hyperedge h_r , which is “guessed” by the algorithm at step (2)—hence, the algorithm is non-deterministic. Given h_r , step (3) is responsible for computing M_r according to Definition 3.7. Then, for all options C_r that are available to the Robber after the move (h_r, M_r) , the algorithm checks recursively at step (4) whether there is a winning strategy for the Captain. The algorithm returns TRUE if, and only if, all such recursive calls succeed. We next show that the algorithm is correct and that it can be implemented to take polynomial time.

Concerning the correctness, due to its non-deterministic nature, it is easily seen that, by getting rid of step (1), it returns TRUE if, and only if, the Captain has a greedy winning strategy in the game played on $(\mathcal{H}_1, \mathcal{H}_2)$ (which we assume to be “visible” by the function at a every call, to avoid a longer signature). Moreover, we claim that the check performed at step (1) cannot lead to a wrong FALSE output. Indeed, just observe that the number of recursive calls is bounded by the number of all distinct configurations, which is $\text{MaxGreedyStrat}(\mathcal{H}_1, \mathcal{H}_2)$ at most, by Lemma 3.9. Therefore, if the recursion level i exceeds this threshold, then we can safely answer FALSE.

Let us now focus on the running time. We know that GREEDYWINNINGSTRATEGY can be implemented on an alternating Turing machine \mathcal{M}_G , whose existential steps correspond to the guess statements at step 2, while universal steps are used for checking that the conditions at step 4 are satisfied by all the relevant components. In addition, by indexing the various data structures and by referring each component via one point

contained in it (selected through any fixed criterium), the machine can be implemented to use logarithmic many bits on its worktape.

For instance, recall from the proof of Lemma 3.9 that every configuration is identified by at most four elements of the form (h_p, h_r, X_p, X_r) with $h_p, h_r \in \text{edges}(\mathcal{H}_2)$ and $X_p, X_r \in \text{nodes}(\mathcal{H}_1)$. Therefore, any configuration may be encoded by (at most) four indexes whose maximum size is $\log \max\{|\text{edges}(\mathcal{H}_2)|, |\text{nodes}(\mathcal{H}_1)|\}$. Moreover, the check at step (1) ensures that the length of each branch of the computation tree of \mathcal{M}_G is finite, and actually bounded by a polynomial in the size of the input. For the sake of completeness, observe that all subtasks in the function, such as computing connected components and the like, are easily implementable in nondeterministic logspace, so that such tasks just correspond to further (polynomially-bounded) branches of the computation tree of \mathcal{M}_G . Thus, GREEDYWINNINGSTRATEGY may be implemented as a *log-space* alternating Turing machine, which immediately entails the result, because Alternating Logspace is equal to Polynomial Time [15]. \square

It is well known that an alternating Turing machine \mathcal{M}_G can be simulated by a standard machine in polynomial time. First, compute the polynomially-many possible instant descriptions (IDs) of the machine, and build a graph representing the possible connections between any pair of IDs, according to its transition relation. Then, evaluate this graph along some topological ordering as follows. Mark all IDs without outgoing arcs associated with final accepting states; then mark all IDs associated with existential states having a marked successor, or associated with universal states, and whose successors are all marked. Then, the machine \mathcal{M}_G accepts its input if, and only if, the starting ID is marked. Moreover, the subgraph induced by the marked nodes encodes its accepting computations.

Moreover, from such a marked graph it is straightforward to compute the strategy graph of a greedy winning strategy, because IDs associated with (children of) existential states encode the possible choices of the Captain.⁵ Just visit the graph starting from the initial configuration, but for each ID associated with an existential state, select one child to be visited arbitrarily (all choices are marked and hence accepting).

Corollary 3.11. *The strategy graph of a greedy winning strategy (if any) in the Robber and Captain game is computable in polynomial time.*

4. Larger Islands of Tractability

From the previous section (see Theorem 3.4 and Example 3.8), we know that monotone winning strategies for the Captain in the game over $(\mathcal{H}_1, \mathcal{H}_2)$ are associated with tree projections of \mathcal{H}_1 w.r.t. \mathcal{H}_2 , and that in some cases it is possible that there is no monotone winning greedy strategy, although monotone winning strategies (non-greedy) exist. In this section, we show that from *any* (possibly non-monotone) greedy winning strategy a tree projection can be still computed in polynomial time. The key fact here is

⁵For the sake of completeness note that, by using these ideas, one might also provide a direct dynamic programming algorithm to compute a strategy graph by using a bipartite graph representing all possible configurations and positions of the Robber and Captain game. However, we find the non-deterministic function GREEDYWINNINGSTRATEGY more elegant and easy to present.

that any non-monotone greedy strategy can be converted into a monotone one, though not a greedy one in general. Based on this observation, a larger island of tractability for tree projections will be eventually singled out.

4.1. Nice Strategies and Greedy Tree Projections

To establish our results, it is useful to consider a special form of strategies that we call *nice* (for they remind the notion of nice tree decompositions of graphs), where at every configuration the Captain first removes those cops that are no longer in the frontier.

Formally, σ is a *nice* strategy if $\sigma(h_p, M_p, C_p) = (h_p, \partial C_p)$, whenever $\partial C_p \subset M_p$. Because such inactive cops play no role in the Robber and Captain game, a winning nice strategy exists if (and only if) there exists a winning strategy, and the same holds for greedy strategies. Just note that restricting the cops to the border of C_p is a legal choice in greedy strategies (it corresponds to the selection of the same squad $h_r = h_p$ before attacking the robber in the component C_p with some further squad). Clearly enough, such a nice strategy can be computed in polynomial time from any given strategy. Also, if desired, the polynomial time algorithm for computing a greedy strategy may be easily adapted to compute directly a winning nice greedy strategy (if any).

Example 4.1. Consider again the setting discussed in Example 3.2 and illustrated in Figure 4. Note that the strategy σ is not nice. Indeed, Figure 7 reports the strategy graph associated with a strategy σ_n that is nice and that is obtained from σ by just explicitly adding the configurations where the Captain has to remove the cops that are no longer in the frontier. \triangleleft

The reason for introducing nice strategies is that they admit a compact representation. First, given any configuration (h_p, M_p, C_p) and a Captain's choice M_r , the $[(h_p, M_p, C_p), M_r]$ -options for the Robber are determined by C_p and M_r only, because ∂C_p is computable from C_p . Therefore, we use hereafter the simplified notation $[C_p, M_r]$ -option to refer to this set of $[M_r]$ -components. Moreover, in place of the strategy graph, we can use a *component graph*, defined as follows.

Definition 4.2. Let $(\mathcal{H}_1, \mathcal{H}_2)$ be a pair of hypergraphs. Let $G = (N, A)$ be a directed graph whose nodes are pairs of the form (h_p, C_p) , where $h_p \in \text{edges}(\mathcal{H}_2)$, and C_p is either the emptyset or a $[\partial C_p]$ -component of \mathcal{H}_1 such that $\partial C_p \subseteq h_p$. Then, we say that G is a *component graph* if it meets the following conditions:

- (1) There is a *root* node $(\emptyset, \text{nodes}(\mathcal{H}_1)) \in N$ that is the only node without incoming arcs.
- (2) Each node $(h_p, C_p) \in N$, with $C_p \neq \emptyset$, has outgoing arcs to $m \geq 0$ nodes $(h_r, \bar{C}_1), \dots, (h_r, \bar{C}_m)$ such that, if M_r is the set $\bigcup_{j=1}^m \partial \bar{C}_j \cup (C_p \setminus \bigcup_{j=1}^m \bar{C}_j)$, it holds that $M_r \subseteq h_r$ and the $[C_p, M_r]$ -options are the components $\bar{C}_1, \dots, \bar{C}_m$.
- (3) Each node $(h_p, C_p) \in N$ has an outgoing arc to (h_r, \emptyset) if $C_p \subseteq h_r$.

Note that every nice strategy σ is encoded by the component graph $G_c(\sigma) = (N, A)$ defined as follows. There is a node (h_p, C_p) (resp., (h_p, \emptyset)) in N if there is a configuration $(h_p, \partial C_p, C_p)$ in the domain of σ (resp., a capture configuration $(h_p, \partial C_p, \emptyset)$ induced by σ). There is an arc in A from a node (h_p, C_p) to a node (h_r, C_r) if there is an arc from (h_p, M_p, C_p) to (h_r, M_r, C_r) in the strategy graph $G(\sigma)$. No more nodes and arcs occur in

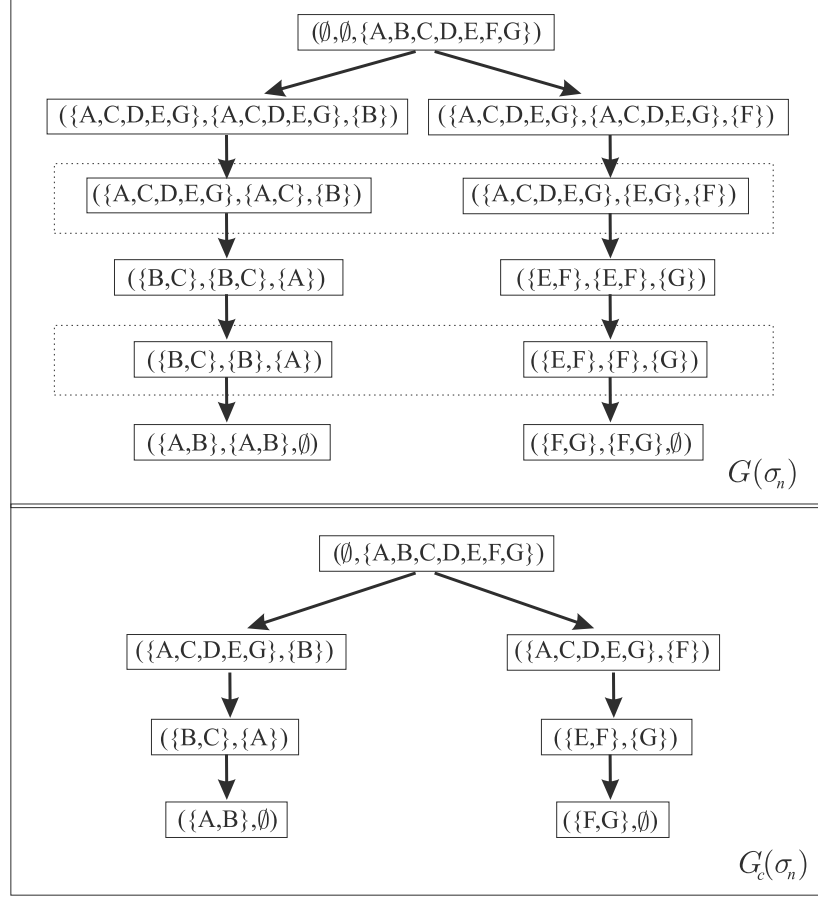


Figure 7: The strategy and component graphs for the nice strategy σ_n in Example 4.1.

N and A , respectively. For instance, the graph depicted on the bottom part of Figure 4 is the component graph associated with the nice strategy σ_n of Example 4.1.

Conversely, any component graph G encodes a nice strategy σ_G , via the following procedure. Associate the root $(\emptyset, nodes(\mathcal{H}_1))$ with the initial configuration $(\emptyset, \emptyset, nodes(\mathcal{H}_1))$. Inductively, assume that a node (h_p, C_p) is associated with a configuration (h_p, M_p, C_p) , and that $(h_r, \bar{C}_1), \dots, (h_r, \bar{C}_m)$ are the labels of the nodes having an incoming arc from (h_p, M_p, C_p) . Let $M_r = \bigcup_{j=1}^m \partial \bar{C}_j \cup (C_p \setminus \bigcup_{j=1}^m \bar{C}_j)$, with $M_r \subseteq h_r$. Then, define $\sigma_G(h_p, M_p, C_p) = (h_r, M_r)$, and define $\sigma_G(h_r, M_r, \bar{C}_j) = (h_r, \partial \bar{C}_j)$, with $j \in \{1, \dots, m\}$, in the case where $\partial \bar{C}_j \subset M_r$.

Theorem 4.3. *A tree projection of \mathcal{H}_1 w.r.t. \mathcal{H}_2 can be computed in polynomial time if the Captain has a greedy winning strategy on $(\mathcal{H}_1, \mathcal{H}_2)$.*

Proof. By Theorem 3.10, we can decide in polynomial time whether a winning greedy strategy for the Captain in the game played on $(\mathcal{H}_1, \mathcal{H}_2)$ exists or not. In the negative case, we are done. Otherwise, we compute in polynomial time a winning nice greedy

strategy σ (or turn a given strategy into a nice one), and we subsequently build its component graph $G_c(\sigma)$. Let us now make a copy $G' = (N', A')$ of $G_c(\sigma)$, and note that G' is a directed acyclic graph, because it encodes a winning strategy.

The line of the proof is to show that we can incrementally modify G' , until we end up with a component graph still encoding a nice winning strategy that is however monotone; the result then follows because a tree projection can be computed in polynomial time from a monotone winning strategy [38]. We process G' from the leaves to the root, according to any of its topological orderings. Whenever we process a node v_j of G' that is associated with a non-monotone move, we shall eventually apply a local transformation to G' , discussed in the steps (i)–(iv) detailed below and illustrated in Example 4.4. In particular, we shall show that the updated graph is still a component graph encoding a nice winning strategy. Moreover, we shall observe that, after the execution of steps (i)–(iv), the game encoded in the modified graph and starting at v_j is monotone. Hence, after the root is processed, we end up with a monotone winning strategy. The proof will be completed by observing that the number of such transformations is polynomially bounded.

Let us now formalize the approach sketched above. Let $\vec{N} = v_1, \dots, v_{|N'|}$ be the topologically ordered sequence of the nodes of G' , where the nodes without outgoing arcs, called leaves, are in the first positions, and the node without incoming arcs, its root, is at the last position. Note that leaves correspond to capture configurations for the robber, while the root $v_{|N'|} = (\emptyset, \text{nodes}(\mathcal{H}_1))$ is associated with the starting configuration $(\emptyset, \emptyset, \text{nodes}(\mathcal{H}_1))$ of the game. Moreover, if $(v, v') \in A'$, the node v is said to be a parent of v' , while v' is said to be a child of v . Then, we start modifying the graph G' , by navigating the sequence \vec{N} using an index j , as discussed below.

Starting with $j = 1$, while $j < |N'|$, consider the current node v_j in the sequence, associated with a configuration (h_j, M_j, C_j) (initially, the first leaf) in the domain of $\sigma_{G'}$. If every child of v_j is labeled by some (h'', C'') with $C'' \subseteq C_j$, then let index $j := j + 1$ and continue the “while” loop, or stop and output the current graph G' if v_j is the root. Otherwise, let v_s be a child of v_j labeled by $(h_s, C_s) \in N'$ such that $C_s \not\subseteq C_j$, and associated with the configuration (h_s, M_s, C_s) . That is, $\sigma_{G'}(h_j, M_j, C_j) = (h_s, M_s)$ is a non-monotone move. Then, take any parent v_p of v_j , and let (h_p, M_p, C_p) the configuration associated with v_p (whose label is thus (h_p, C_p)). Modify the graph so that $\sigma_{G'}(h_p, M_p, C_p) = (h_j, M'_j)$, where $M'_j = M_j \setminus \text{ED}(v_j, M_s)$. In particular, let C'_j be the $[M'_j]$ -component that properly includes C_j , and for which thus $C_p \cup C'_j$ is $[M_p \cap M'_j]$ -connected. Then, the modified component graph will also encode the choice $\sigma_{G'}(h_j, M'_j, C'_j) = (h_j, \partial C'_j)$ if $\partial C'_j \subset M'_j$, and $\sigma_{G'}(h_j, \partial C'_j, C'_j) = (h_s, M_s)$. The transformation of the graph is as follows:

- (i) Add a node v'_j labeled by (h_j, C'_j) to N' and to the sequence \vec{N} in the position before v_j , and add to A' an arc from v'_j to each child of v_j , i.e., to nodes labeled by (h_s, C'') , for each $[C'_j, M_s]$ -option C'' .
- (ii) Remove from A' all outgoing arcs of v_p to nodes whose labels do not contain $[C_p, M'_j]$ -options (in particular, the arc towards v_j is removed).
- (iii) Add to A' an arc from v_p to v'_j .
- (iv) Remove from N' any node different from the root which is left without incoming

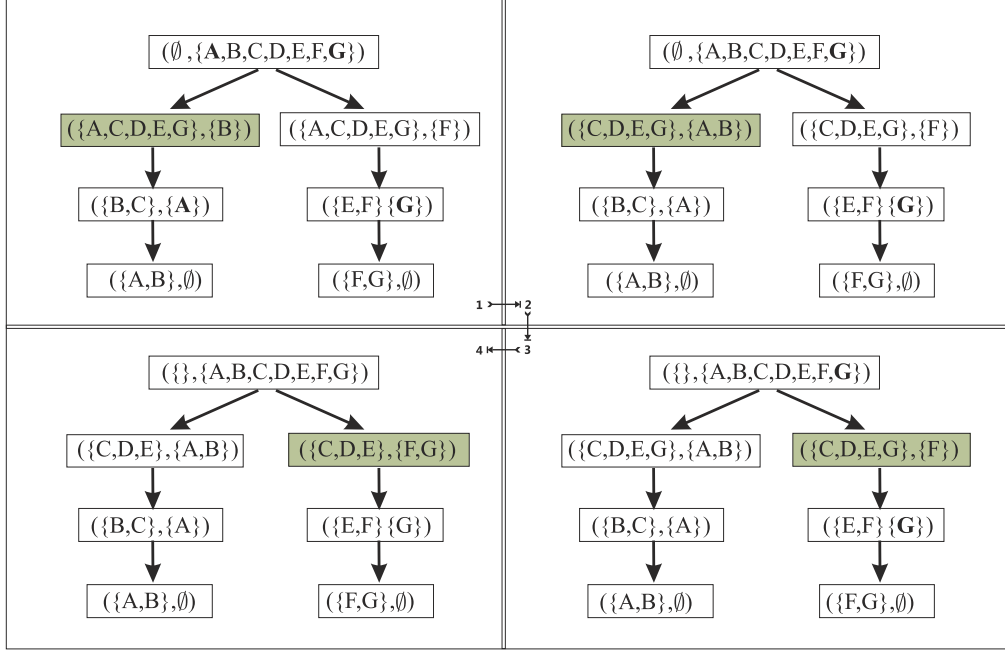


Figure 8: Illustration of the algorithm in the proof of Theorem 4.3.

arcs, and continue the “while” loop considering again node v_j , or the next available node in \vec{N} if v_j has been removed by N' .

Example 4.4. The application of the above procedure to the nice strategy σ_n discussed in Example 4.1 is illustrated in Figure 8. Note that two non-monotone moves are removed in total. Note that, at the end of the transformation, we get a component graph encoding precisely the monotone strategy $\bar{\sigma}$, whose strategy graph is illustrated in Figure 5. \triangleleft

First observe that every iteration of the loop at step 1 above, precisely implements on the graph G' the transformation (of the non-monotone strategy encoded by G') described by Expression (1), and whose properties are described by Lemma 3.6. In more detail, with these properties in mind, by executing steps (i)–(iii) we replace the Captain’s choice (h_j, M_j) at (h_p, M_p, C_p) by the new choice (h_j, M'_j) , and we get the following situation: (a) Because of the new choice M'_j , only one new $[C_p, M'_j]$ -option is available to the robber, that is, the $[M'_j]$ -component C'_j properly including the $[M_j]$ -component C_j . As a consequence, at step (i) the one node v'_j corresponding to this component is added to N' . (b) The $[C'_j, M_s]$ -options are the same as the $[C_j, M_s]$ -options, so that the outgoing arcs of v'_j will be the same as the node v_j . That is, we keep the same winning strategy as before, as the Robber’s options after the Captain’s choice M_s are the same as before (and hence the Captain knows how to successfully attack them). (c) The set of $[C_p, M'_j]$ -options, with the exception of the new C'_j , are a subset of the $[C_p, M_j]$ -options. In fact, some components may collapse after the new choice of the Captain. Then, at step (iv), we

remove the nodes associated with $[C_p, M_j]$ -options that are now left without incoming arcs. For instance, it is possible that we delete v_j if v_p was its only parent, or it is possible that we delete some nodes associated with collapsed components. Note that the new graph G' obtained from these steps is still a component graph, hence it encodes a (new) nice strategy $\sigma_{G'}$.

Therefore, Lemma 3.6 entails that, after each iteration and thus after the entire procedure, the strategy $\sigma_{G'}$ is a winning strategy. We claim that it is actually a monotone winning strategy, by a simple inductive argument: if v_j is the current node, after the execution of steps (i)–(iv), $\sigma_{G'}$ is a monotone winning strategy for the game starting at the configuration v_j . Then, the claim follows because, for $j = |N'|$, it means that $\sigma_{G'}$ is a monotone winning strategy for the whole game starting at the root.

The base case is when the algorithm starts at $j = 1$, and hence the statement holds because the first position in \vec{N} is occupied by some leaf, which is a capture configuration of the winning strategy. Now assume that the statement holds for $j - 1$, and consider the execution of the above procedure on node v_j . Note that the proposed transformation deals with just one (possibly new) component C'_j instead of the strictly smaller C_j ; everything else in the strategy does not change, in particular no node preceding v_j in the topological order is affected by the transformation. Then, the monotonicity of the strategy on the game starting at v_j immediately follows from the induction hypothesis and from Lemma 3.6.(1), which says that $\text{ED}(v'_j, M_s) = \emptyset$ and hence that this move is monotone, so that $C'' \subseteq C'_j$, for each $[C'_j, M_{i+1}]$ -option C'' .

Because each iteration is feasible in polynomial time, it just remains to show that the whole procedure requires at most polynomially many iterations. To this end, note that whenever some node v_j encodes a non-monotone move, one node v'_j is added to N' for each parent v_p of v_j . Indeed, the node v_j is considered again after the first iteration where it was evaluated, if it still has incoming arcs (see step (iv)). However, after steps (i)–(iv), $\sigma_{G'}$ is a monotone winning strategy for the game starting at the new configuration v'_j . Therefore, no new node will be subject to further transformations in subsequent iterations along the given topological ordering of N' . It follows that the number of iterations of the described procedure is bounded by $\text{nodes}(G_c(\sigma)) \times \text{MaxIn}$, where MaxIn is the largest in-degree over the nodes of $G_c(\sigma)$. Thus, the number of iterations is bounded by a polynomial in the size of the strategy graph of the greedy winning strategy, which is in its turn polynomial in the size of $(\mathcal{H}_1, \mathcal{H}_2)$.

Finally, from the monotone winning strategy $\sigma_{G'}$ encoded by the output G' of the above procedure, a tree projection \mathcal{H}_a of $(\mathcal{H}_1, \mathcal{H}_2)$ is available. Just define $\text{nodes}(\mathcal{H}_a) = \text{nodes}(\mathcal{H}_1)$ and $\text{edges}(\mathcal{H}_a) = \{M \mid \sigma_{G'}(v) = (h, M) \text{ for some configuration } v \text{ in the domain of } \sigma_{G'}\}$. See [38], for more detail about such a relationship between monotone strategies and tree projections. \square

With the above result in place, let \mathcal{C}_{gtp} denote the class of all pairs (Q, \mathcal{V}) such that there exists a greedy winning strategy σ for the Captain in the Robber and Captain game on $(\mathcal{H}_Q, \mathcal{H}_\mathcal{V})$. As shown in the proof of Theorem 4.3, based on σ a tree projection of \mathcal{H}_Q w.r.t. $\mathcal{H}_\mathcal{V}$, which we call *greedy tree projection*, can be computed in polynomial time. Therefore, the following is established.

Corollary 4.5. *\mathcal{C}_{gtp} is an island of tractability.*

4.2. Captain vs Marshal

A class of tractable pairs related to our class \mathcal{C}_{gtp} has been defined in [4] in terms of the *Robber and Marshal game* played by one Marshal and the Robber on the hypergraphs $(\mathcal{H}_1, \mathcal{H}_2)$. This game has been originally defined on a single hypergraph to characterize hypertree decompositions [35], and its natural extension to pairs of hypergraphs has been defined and studied in [4].

The game is as follows. The Marshal may control one hyperedge of \mathcal{H}_2 , at each step. The Robber stands on a node and can run at great speed along hyperedges of \mathcal{H}_1 ; however, (s)he is not permitted to run through a node that is controlled by the Marshal. Thus, a *configuration* is a pair (h, C) , where h is the hyperedge controlled by the Marshal, and C is an $[h]$ -component where the Robber stands. Let (h_p, C_p) be a configuration. This is a *capture configuration*, where the Marshal wins, if $C_p \subseteq h_p$. Otherwise, the Marshal moves to another hyperedge $h_r \in \text{edges}(\mathcal{H}_2)$; while (s)he moves, the Robber may run through those nodes that are left by the Marshal or not yet occupied. Thus, the Robber selects an $[h_r]$ -component C_r such that $C_r \cup C_p$ is $[h_p \cap h_r]$ -connected. We say that the Marshal has a *winning strategy* if, starting from the initial configuration (\emptyset, \mathcal{N}) , (s)he may end up the game in a capture position, no matter of the Robber's moves. A winning strategy is *monotone* if the Marshal may monotonically shrink the set of nodes where the Robber stands.

Because only nodes in the frontier are actually used at each step in the monotone Robber and Marshal game, this game and the monotone variants of the Robber and Captain game clearly define the same hypergraph properties.

Fact 4.6. *The following are equivalent:*

- (1) *There is a monotone winning strategy for the Marshal in the Robber and Marshal game on $(\mathcal{H}_1, \mathcal{H}_2)$.*
- (2) *There is a monotone winning greedy-strategy for the Captain in the Robber and Captain game on $(\mathcal{H}_1, \mathcal{H}_2)$.*

Let \mathcal{C}_{rm} denote the class of all pairs (Q, \mathcal{V}) such that there exists a monotone winning strategy for the Marshal on $(\mathcal{H}_Q, \mathcal{H}_{\mathcal{V}})$. From the results in [4, 3], \mathcal{C}_{rm} is an island of tractability as well. However, the set of tractable instances identified by greedy winning strategies in the Robber and Captain game properly includes this class. The reason is that greedy winning strategies are allowed to be non-monotone.

Theorem 4.7. $\mathcal{C}_{rm} \subset \mathcal{C}_{gtp}$.

Proof. Because greedy strategies are not required to be monotone, $\mathcal{C}_{rm} \subseteq \mathcal{C}_{gtp}$ follows from Fact 4.6. For the proper inclusion, just consider again Example 3.8. The pair of hypergraphs shown in Figure 4 is such that the Marshal has no monotone winning strategy, while the Captain has a (non-monotone) winning greedy strategy.⁶ \square

⁶This example is in fact inspired by a similar simpler pair of hypergraphs where no monotone strategy for the Marshal exists, described in [4].

For completeness, recall that the non-monotone variant of the Marshal and Robber game is instead too powerful to be useful. Indeed, there are pairs of hypergraphs where the Marshal has a non-monotone winning strategy but no tree projection exists. We refer the interested reader to [4] for more detail about the monotonicity gap in the Robber and Marshal game.

5. Applications

In this section, we explore two applications of the results derived about greedy tree projections. In particular, we first move from the general setting of tree projections to analyze specific decomposition methods, and we then focus on tree projections for queries to be answered over databases whose relations have “small” arities.

5.1. Greedy Hypertree Decompositions and Further Greedy Methods

The tractability result about the general case of greedy tree projections can be immediately applied to every structural decomposition method, in order to get new tractable variants of these methods. To carry out the elaborations, observe that any structural decomposition method DM can be viewed as a method associating a set \mathcal{V} of views to any given query Q . Indeed, the decompositions of Q according to DM are precisely tree projections of \mathcal{H}_Q w.r.t. $\mathcal{H}_{\mathcal{V}}$.

Given this correspondence, it is then natural to consider the greedy variant of any structural decomposition method DM , denoted by *greedy-DM*, whose associated decompositions are the *greedy* tree projections of \mathcal{H}_Q w.r.t. $\mathcal{H}_{\mathcal{V}}$. From Corollary 4.5, every decomposition method, possibly an intractable one such as the generalized hypertree decomposition method, defines an island of tractability by means of its greedy variant.

Fact 5.1. *Let DM be a structural decomposition method and let *greedy-DM* be its greedy variant. Then, the class of all queries having a *greedy-DM* decomposition is recognizable in polynomial time, and every query in the class may be evaluated in polynomial time over any given database.*

We next focus on the greedy variant of the method based on generalized hypertree decompositions. Let $k \geq 1$. Recall from Section 2 that the width- k generalized hypertree decompositions of a query Q are the tree projections of $(\mathcal{H}_Q, \mathcal{H}_Q^k)$. Indeed, we are considering one distinct view over each set of variables that can be covered by at most k query-atoms.

Definition 5.2. A width- k *greedy hypertree-decomposition* (we omit “generalized”, for short) of a conjunctive query Q is any greedy tree projection of $(\mathcal{H}_Q, \mathcal{H}_Q^k)$. Accordingly, the *greedy (generalized) hypertree-width* of Q , denoted by *gr-hw*, is the smallest k such that Q has a greedy hypertree decomposition of width k .

This greedy variant provides a new tractable approximation of the (intractable) notion of generalized hypertree decomposition, which is better than (standard) hypertree decompositions.

Fact 5.3. *For any query Q , $ghw(Q) \leq gr-hw(Q) \leq hw(Q)$ holds. Moreover, there are queries Q for which $gr-hw(Q) < hw(Q)$, even for $gr-hw(Q) = 2$.*

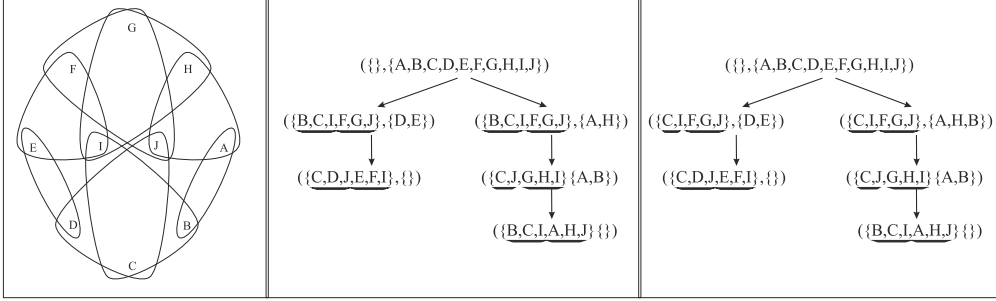


Figure 9: Examples in the proof of Fact 5.3.

Proof. The first relationship is immediate: in the first inequality we use the fact that greedy hypertree decompositions are a special case of generalized hypertree decompositions, while the second inequality holds because the notion of hypertree decomposition is characterized by the monotone Robber and Marshals game, played on \mathcal{H}_Q by a Robber and k Marshals [35]. This game is equivalent to play the monotone game with one Marshal on the pair of hypergraphs $(\mathcal{H}_Q, \mathcal{H}_Q^k)$, which is the same as playing the monotone Robber and Captain game.

For the strict upper bound $gr-hw(Q) < hw(Q)$, consider the query Q_0 , taken from [17, 36], whose hypergraph \mathcal{H}_{Q_0} is depicted in the left part of Figure 9. For this query, it is shown in [36] that $hw(Q_0) = 3$ and $ghw(Q_0) = 2$. However, $gr-hw(Q_0) = 2$ holds. Indeed, there is a winning greedy strategy for the Captain in the game played on $(\mathcal{H}_{Q_0}, \mathcal{H}_{Q_0}^2)$, as shown in the central part of Figure 9, and thus there exists a greedy tree projection of \mathcal{H}_{Q_0} w.r.t. $\mathcal{H}_{Q_0}^2$.

In the figure, the set of selected cops at each step is underlined in such a way that the reader may identify the original pair of hyperedges from \mathcal{H}_{Q_0} that forms the chosen squad in $\mathcal{H}_{Q_0}^2$. Note that the strategy is non-monotone, as it is witnessed by the right branch where the Robber can return on the node B . However, by using the construction in Theorem 4.3, it can be turned into a monotone (while not greedy) one, by removing the escape door B in the first move of the Captain (see the right part of the figure). From the monotone strategy, we immediately get the desired tree projection. \square

More general examples are given by the *subedge-based* decomposition methods, defined in [36]. Recall that a subedge-method DM is based on a function f associating with each integer $k \geq 1$ and each hypergraph $\mathcal{H}_Q = (V, E)$ of some query Q a set $f(\mathcal{H}_Q, k)$ of subedges of \mathcal{H}_Q , that is, a set of subsets of hyperedges in E . Moreover, the set of width- k DM-decompositions of Q can be obtained as follows: (1) obtain a hypertree decomposition HD of $\mathcal{H}_f = (V, E \cup f(\mathcal{H}, k))$, and (2) convert HD into a generalized hypertree decomposition of \mathcal{H}_Q by replacing each subedge $h \in f(\mathcal{H}_Q, k) \setminus E$ occurring in HD by some hyperedge $h' \in E$ such that $h \subseteq h'$ (which exists because h is a subedge).

Because such a method is based on width- k hypertree decompositions, in the tree projection framework it can be recast as follows. A width- k DM-decomposition is any tree decomposition of \mathcal{H}_Q w.r.t. \mathcal{H}_f^k associated with some monotone winning strategy of the Robber and Marshal game on this pair of hypergraphs. On the other hand, according to its greedy variant *greedy-DM*, the width- k decompositions are the greedy tree projections

of \mathcal{H}_Q w.r.t. \mathcal{H}_f^k . It follows that the greedy variant of this method is more powerful.

Fact 5.4. *Let DM be any subedge-based decomposition method. Let $k \geq 1$ and let Q be a query. Then, a width- k DM-decomposition of Q exists only if a width- k greedy-DM-decomposition of Q exists. The converse does not hold, in general.*

Proof. The first entailment follows from Theorem 4.7. The fact that the converse does not hold in general, follows from Fact 5.3, because the hypertree decomposition method is a subedge-based method (based on the function $f(\mathcal{H}_Q, k) = \emptyset$). \square

This is a remarkable result, as in [36] some examples of subedge-based decomposition methods, such as the *component hypertree decompositions*, are shown to generalize most previous proposals of tractable structural decomposition methods, such as hypertree and spread-cut decompositions (in fact, all of them, but the approximation of fractional hypertree decomposition, later introduced in [49]). From Fact 5.4, their greedy variants are even more powerful.

5.2. Tractability over Small Arity Structures

We now consider the case of relational structures having small arity, which is a relevant special case in real-world applications. In fact, observe that any variable that is not involved in any join operation in a conjunctive query (that is, any variable that occurs in one atom only) is irrelevant and may be projected out in a preprocessing phase. It follows that the *effective arity* to be considered in our structural techniques is actually determined by the largest number of variables that any atom has in common with other atoms (i.e., those variables involved in join operations), independently of the arity of the relations in the original database schema. This number is often small, in practice.⁷

Therefore, it is interesting to investigate whether the general problem of computing a tree projection of a pair of hypergraphs is any easier in the case of small arity structures (for the sake of presentation, we just consider here the standard structure arity, leaving to the interested reader the straightforward extension to the above mentioned “effective arity”). We next show that the problem is indeed in polynomial-time for bounded-arity structures, and it is moreover *fixed-parameter tractable (FPT)*, if the arity is used as a parameter of the problem. This is not difficult to prove, but it was never stated before (as far as we know), and we believe it is important to pinpoint this tractability result.

Recall that a problem is FPT if there is an algorithm that solves the problem in *fixed-parameter polynomial-time*, that is, with a cost $f(k)O(n^{O(1)})$, for some computable function f that is applied to the parameter k only. In other words, this algorithm not only runs in polynomial time if k is bounded by a fixed number, but it also exhibits a “nice” dependency on the parameter, because k is not in the exponent of the input size n . Let p -TP be the problem of computing a tree projection of \mathcal{H}_Q w.r.t. \mathcal{H}_V , for a given pair (Q, \mathcal{V}) , parameterized by the maximum arity of the relations occurring in (Q, \mathcal{V}) .

Theorem 5.5. *The problem p -TP is fixed-parameter tractable.*

⁷In fact, it is easy to further generalize this line of reasoning, by considering as “effective arity” the maximum cardinality over the hyperedges in the GYO-reduct of \mathcal{H}_Q . (Recall that the GYO reduct of a hypergraph is obtained by iteratively removing nodes that occur in one hyperedge only and hyperedges included in other hyperedges, until no further removal is possible—see, e.g., [56].)

Proof. Let (Q, \mathcal{V}) be an input pair for p -TP, let $(\mathcal{H}_Q, \mathcal{H}_\mathcal{V})$ be the pair of associated hypergraphs, and let k be the parameter.

Compute the simplicial version \mathcal{H}_s of the hypergraph $\mathcal{H}_\mathcal{V}$, that is, the hypergraph having the same set of nodes as $\mathcal{H}_\mathcal{V}$, and where $\text{edges}(\mathcal{H}_s) = \{h' \neq \emptyset \mid h' \subseteq h, h \in \text{edges}(\mathcal{H}_\mathcal{V})\}$. Therefore, $\text{edges}(\mathcal{H}_s)$ contains all subsets of every hyperedge of $\mathcal{H}_\mathcal{V}$. Clearly, \mathcal{H}_s can be computed in time $O(2^k \times |\text{edges}(\mathcal{H}_\mathcal{V})|)$, and the tree projections of $(\mathcal{H}_Q, \mathcal{H}_\mathcal{V})$ are the same as the tree projections of $(\mathcal{H}_Q, \mathcal{H}_s)$. To conclude, observe that any tree projection of the latter pair can be computed in polynomial-time by Theorem 4.3 and the fact that, having a squad for every possible set of cops in any squad/hyperedge of $\mathcal{H}_\mathcal{V}$, the greedy strategies in the game Robber and Captain on $(\mathcal{H}_Q, \mathcal{H}_s)$ are precisely the (unrestricted) strategies in the Robber and Captain game on $(\mathcal{H}_Q, \mathcal{H}_\mathcal{V})$, which characterize the tree projections of $(\mathcal{H}_Q, \mathcal{H}_\mathcal{V})$.⁸ \square

The above tractability result is smoothly inherited by all structural decomposition methods DM such that the arity of the views is $O(f(k))$ for some computable function f that does not depend on the size of the input. For instance, this is the case for the methods based on bounded (generalized hyper)tree decompositions, but not for fractional hypertree decompositions. In particular, if w is the fixed maximum width for a class of queries having bounded generalized hypertree width, the maximum arity of the computed views is $w \times k$. Thus, if $p\text{-ghw}_w$ denotes the problem of computing a width- w generalized hypertree decomposition of a query, parameterized by the maximum arity of the query atoms, we immediately get the following result.

Corollary 5.6. *The problem $p\text{-ghw}_w$ is fixed-parameter tractable.*

6. From Theory to Practice

Many recent works are using structural methods based on the computation of a tree projection of the given instance, such as generalized hypertree decompositions or fractional hypertree decompositions, for answering queries to relational databases or solving constraint satisfaction problems (CSPs), where constraints are represented as finite relations encoding the allowed tuples of values. Moreover, structural methods find applications in game theory and combinatorial auctions, as well as in other fields (see [32] for more information and references on these applications, with a focus on hypertree decompositions). This is quite natural because we are actually using a basic hypergraph-theoretic notion that, in principle, may be useful in any application where acyclic instances are easy to deal with. In the rest of the section, we discuss some of these applications.

6.1. Using Tree Projections

Tree projections represent transformations from a given problem to its acyclic variant. For instance, consider a conjunctive query Q over a database instance DB, and assume that its associated hypergraph \mathcal{H}_Q is cyclic. Given a set of available views \mathcal{V} , any tree

⁸Note that the same relationship holds for the monotone strategies and, hence, for the Marshal's strategies in the Robber and Marshal game over the pair $(\mathcal{H}_Q, \mathcal{H}_s)$, as observed by Adler [3].

projection \mathcal{H}_a of \mathcal{H}_Q with respect to \mathcal{H}_V can be used to obtain an acyclic query Q' on a database DB' that is equivalent to Q on DB :⁹ For each hyperedge h of \mathcal{H}_a , compute a fresh atom such that its set of variables is h and its relation is obtained by projecting on h the relation associated with any view $w \in \mathcal{V}$ whose set of nodes includes h (such a view exists by definition of tree projection). This immediately provides a polynomial-time upper bound on the running time of answering the query. Let r be the size of the largest relation associated with the views in \mathcal{V} , and let m be the number of hyperedges of \mathcal{H}_a , which is known to be bounded by the number of variables (in the so-called normal form tree projection [38]). The above transformation is feasible in $O(m \cdot r)$, with each relation in the new database DB' having at most r tuples. Let $r' \leq r$ be the actual size of the largest relation of the database DB' . The overall complexity immediately follows by adding the cost of evaluating the new acyclic instance (e.g., by Yannakakis's algorithm [57]), which dominates the overall cost: the worst-case upper bound is $O(m \cdot (r' + s) \cdot \log(r' + s))$ time and $O(m \cdot (r' + s))$ space, where s is the size of the output.

As a further example of applications of methods based on tree projections, we mention the EmptyHeaded relational engine that uses generalized hypertree decompositions in its query planner [2]. In [45], similar techniques based on structural decompositions have been used to guide a flexible caching of intermediate results in the context of computing multiway joins. In [9], a CSP solving technique based on generalized hypertree decompositions and using compressed representations for the relations has been proposed, and its scalability has been assessed over well-known CPS benchmarks.

Finally note that algorithms based on such structural methods can be parallelized, as pointed out in [33]. Generalized hypertree decompositions are indeed used for parallel query answering in the GYM algorithm [6], which is a distributed and generalized version of Yannakakis' algorithm for answering acyclic queries, specifically designed for the MapReduce framework [19].

6.2. Views beyond Structural Decomposition Methods

Consider a pair of hypergraphs $\mathcal{H}_1 \leq \mathcal{H}_2$, of which we want to compute a tree projection \mathcal{H}_a , with $\mathcal{H}_1 \leq \mathcal{H}_a \leq \mathcal{H}_2$. The resource hypergraph \mathcal{H}_2 , whose hyperedges define what we have called views, is completely arbitrary in the general tree projection framework we deal with. As we have seen, specific algorithms for defining views lead to different decomposition methods. We mentioned methods where views are computed in polynomial time (when we talk about islands of tractability), but the tree projection framework is actually much more general.

Views may represent any subproblem that we can use to solve the given instance, or that is already available from previous computations (e.g., materialized views in databases). In some applications, one may relax the polynomial-time constraint and consider instead fixed-parameter tractable computations, for some (application-specific) parameter. In other applications, views may be associated with subproblems that can be solved by using non-structural properties. With this respect, we mention an important line of research in constraint satisfaction, looking at restrictions on the form of specific (fixed) constraint relations, regardless of the structure of constraint scopes, see, e.g., [16].

⁹We actually assume that the relations associated with views are not more restrictive than the original query. This is always the case for the mentioned structural method. For a formal treatment of the general case, see [40].

There are also hybrid approaches, looking at both structure and data [41, 18]. In concrete applications on big databases, the hybrid approach is mandatory: views should be subqueries such that their computation cost is estimated to be low (that is, less than some given threshold), according to information on the actual database instance, such as selectivity of attributes, keys, cardinality of relations, indices, and so on. In [29, 28], a query optimizer taking into account a simple cost model for subquery evaluation, together with views based on the hypertree decomposition method, has been implemented. The optimizer can be put on top of any existing database management system supporting JDBC technology, by transparently replacing its standard optimization module. The results demonstrate a significant gain obtained by using query plans based on hypertree decompositions on queries involving more than two atoms. Further implementations directly inside open-source Database Management Systems are subjects of current work.

6.3. In Practice

There is room for practical applications of the results presented in this paper to improve the efficiency of the above solutions, besides the theoretical interest in providing a better understanding of the difference between the power of general strategies and the power of controlled non-monotonicity in the Robber and Captain game on pair of hypergraphs.

Consider the result on the fixed-parameter tractability of computing a tree projection of \mathcal{H}_1 w.r.t. \mathcal{H}_2 , where the maximum cardinality of the hyperedges in \mathcal{H}_2 (that is, the arity of views, in database terms) is used as the parameter, say k , of the problem. We proved that a tree projection, if any, can be computed in $O(2^k n^{O(1)})$. We believe that this is a useful result because it means that, in all those instances where the number of variables is not large, an effective query optimization (with respect to arbitrary views and hence with respect to any decomposition method) is feasible in reasonable time. Indeed, the computation of the decomposition depends only on hypergraphs (and not on the database) and, unlike other fixed-parameter algorithms, the algorithm described in Theorem 5.5 is “practical,” as there are no (hidden) huge constants and the dependence on the arity parameter is single-exponential. This is of particular interest in database applications, where small queries over huge amount of data are the typical instances. Furthermore, in such a context, the same queries are frequently run over a varying database, so that a good query optimization pays over the time.

We can be even more concrete by focusing on the specific decomposition method based on generalized hypertree decompositions. By Corollary 5.6, a width- w generalized hypertree decomposition of the hypergraph \mathcal{H}_Q of a given query Q can be computed in $O(2^k n^{O(1)})$, where k is the maximum number of variables occurring in any query atom. We next point out that it is very convenient to look for decompositions with the smallest possible widths, which means using the most powerful decomposition methods (that are affordable in the available optimization time). Say n be the combined size of the query Q and the database, and consider the query answering problem parameterized by the generalized hypertree width of Q , say w . It is well known that this problem is not fixed-parameter tractable, which means that (under usual fixed-parameter complexity assumptions) an exponential dependency on the parameter of the form $O(n^{f(w)})$ is unavoidable. It follows that even small savings in the width leads to exponential savings in the query evaluation time (and here n includes the database size). It is worthwhile noting that the same exponential dependency holds if we consider as parameter w the

notion of width associated with other mentioned decomposition methods, in particular the treewidth. We thus argue that investing some time in computing low-width decompositions is very convenient even for queries having small arities. Indeed, $ghw(Q) \leq tw(Q)$ always holds, and for some queries $tw(Q) = k \cdot ghw(Q)$.

The main algorithmic result of this paper, that is, the notion of greedy tree projection and its tractability, is particularly interesting whenever we deal with instances having large hypergraphs. This is often the case in constraint satisfaction problems, where there are instances with hundreds of constraints, for which the computation of a generalized hypertree decomposition having the minimum possible width may not be affordable. Many practical approaches for these applications adapt heuristics developed for the tree decomposition method, or use the notion of hypertree width (see, e.g., [21, 9]). However, as pointed out above, if we are able to find better decompositions, we are guaranteed an exponential-saving in the (worst-case) computation time. In this respect, using greedy tree projections may be a good choice. In particular, the greedy method that we called greedy hypertree decomposition provides always better (or equal) results than hypertree decompositions (and hence than tree decompositions), and it is computable in polynomial time for any fixed, bound on the width.

7. Related Literature on “Cops and Robbers” Games

In this paper we are mainly interested in games defined over hypergraphs or pairs of hypergraphs, such as those studied in [1] (see Section 4.2). We are not aware of many further works of this kind, apart from the *Robber and Army game* [43], which was defined to approximate the notion of fractional hypertree decomposition. This game is indeed a variation of the Robber and Marshals game that characterizes hypertree decompositions, but this time marshals are replaced by a more powerful general, who is in charge of an army of r battalions of soldiers (with r being a rational number). The general may distribute her soldiers on the hyperedges in any arbitrary way (rational allocations are allowed). A node of the hypergraph is blocked (the robber cannot go through that node) if the number of soldiers on all hyperedges that contain this node adds up to the strength of at least one battalion. The game is then played in a monotonic way, like the Robber and Marshals game.

As a matter of fact, all these games, comprising the Robber and Captain game [38] at the core of the present work, can be viewed as variations of the Robber and Cops game defined by Seymour and Thomas over graphs [54], in order to characterize the notion of treewidth. In this game, a number of Cops have to capture a Robber that can run at great speed along the edges of a graph, while being not permitted to run through a node that is controlled by a Cop. In particular, the Cops can move over nodes by using helicopters and, before they land, the Robber is fast and can run through those nodes that are left or not yet occupied before the move is completed. A graph has treewidth bounded by k if, and only if, there is a winning strategy for $k + 1$ Cops in this game [54]. Unlike the Robber and Marshals (or Captain, or Army) game, in the Robber and Cops game, restricting strategies to be monotone does not reduce in any way the power of cops.

By looking at the game defined by Seymour and Thomas, one might naturally wonder what happens if the use of helicopters is not allowed, so that Cops must move along the edges of the graph, precisely as the Robber does. The study of this variant goes back to

the eighties, when it was introduced by Winkler and Nowakowski [51] and independently by Quilliot [52] under the name of the *Cops and Robbers* game. Since then, this game has been the subject of intense study (see the book by Bonato and Nowakowski [14], and the references therein). In particular, in the original formulation, the game proceeds in rounds, each consisting of a Cop turn followed by a Robber turn. In each round, each cop may remain on her current vertex or move to an adjacent vertex, after which the robber likewise chooses to remain in place or move to an adjacent vertex. For this game, several efforts have been spent to characterize the *cop number* of (classes of) graphs, i.e., the minimum number of Cops needed to capture the Robber, regardless of her moves.

Graphs of cop number 1 were characterized already in the above mentioned seminal papers [51, 52]. These graphs are based on a suitable linear ordering of their vertices and can be recognized in polynomial time. Similarly, for any fixed natural number $k > 0$, deciding whether a graph has cop number bounded by k is feasible in polynomial time; indeed, just notice that the number of possible different configurations is $O(n^{k+1})$ for a graph with n vertices. However, for k being part of the input, it has been recently shown that it is EXP-complete to decide if the cop number of a graph does not exceed k [46], hence confirming a long-standing conjecture by Goldstein and Reingold [30]—further complexity results for variants of the game can be found in the works by Fomin et al. [24] and Mamino [48]. For general graphs on n vertices it is known that $\Omega(\sqrt{n})$ cops may be needed, and the celebrated Meyniel’s conjecture [25] states that the cop number of a connected vertex graph is $O(\sqrt{n})$. Moreover, exact or approximate values of the cop number of several classes of graphs have been derived so far, including plan graphs [7], bounded genus graphs [55], and intersection graphs [27], just to name a few.

A variant of the Robber and Cops game discussed above assumes that the Robber is faster than the Cops in that, at each move, she can transverse $s \geq 1$ edges of the graph. This variant has been introduced by Fomin et al. [23], who also showed that computing the cop number in this setting is NP-hard, for every fixed s , even on classes of split graphs. In particular, the case where the Robber has an unbounded speed ($s = \infty$) is very related with the game by Seymour and Thomas characterizing treewidth, except for the use of helicopters. In fact, it turns out that, over planar graphs G , the cop number for this game is $\Theta(tw(G))$ [8]. Further results on this variant can be found in the work by Frieze et al. [26]. Yet another interesting variant of the Cops and Robber game where the robber is invisible has been also studied in the literature (see [20], and the references therein).

While the above games are defined and mainly studied over graphs, the extension of graph games to hypergraph games is sometimes natural. For instance, properties of the hypergraph version of the classical Cops and Robber game in [51, 52] are studied in [10].

8. Conclusion and Future Work

By exploiting a recent game characterization of tree projections, we identified new islands of (structural) tractability based on a greedy version of the powerful non-monotonic strategies in hypergraph games. We show that such greedy strategies can be computed efficiently, and can always guide us towards the computation of useful decomposition trees. In fact, the proposed approach immediately provides larger “greedy” extensions of the most powerful structural decomposition methods defined in the literature.

Furthermore, again using the game-theoretic characterization of tree projections, we pinpoint the fixed-parameter tractability of this notion (and hence of most structural decomposition methods) when the arity is used as the parameter. This models what happens if small arity instances are considered, which often occurs in practice.

We believe that the results presented in the paper may be very useful in real-world applications and we are currently working on direct implementations of the proposed techniques in open-source database management systems. Moreover, note that these results find applications in all those problems that can be solved efficiently on acyclic and quasi-acyclic instances, even outside the Database area we focused on. In particular, our results can be exploited immediately for solving Constraint Satisfaction Problems.

Besides the implementation of efficient algorithms for the computations of greedy hypertree decompositions of large hypergraphs, a research question regards the distance between generalized and greedy hypertree decompositions. From the known relationships with hypertree decompositions, we immediately get that, for any hypergraph \mathcal{H} , $ghw(\mathcal{H}) \leq gr-hw(\mathcal{H}) \leq hw(\mathcal{H}) \leq 3 \cdot ghw(\mathcal{H}) + 1$. However, whether or not these bounds are tight is currently open.

Moreover, it would be interesting to investigate whether the greedy techniques used in this paper for the Robber and Captain game on pair of hypergraphs can be somehow useful for other kinds of (hyper)graphs games, such as those described in Section 7.

References

- [1] I. Adler. Marshals, monotone marshals, and hypertree-width. *Journal of Graph Theory*, 47(4), pp. 275–296, 2004.
- [2] C.R. Aberger, S. Tu, K. Olukotun, and C. Ré. EmptyHeaded: A Relational Engine for Graph Processing. In *Proc. of SIGMOD’16*, 2016.
- [3] I. Adler. Width Functions for Hypertree Decompositions. *PhD Thesis*, University of Freiburg, 2006.
- [4] I. Adler. Tree-Related Widths of Graphs and Hypergraphs. *SIAM Journal Discrete Mathematics*, 22(1), pp. 102–123, 2008.
- [5] I. Adler, G. Gottlob, and M. Grohe. Hypertree-Width and Related Hypergraph Invariants. *European Journal of Combinatorics*, 28, pp. 2167–2181, 2007.
- [6] F.N. Afrati, M. Joglekar, C. Ré, S. Salihoglu, and J.D. Ullman. GYM: A Multiround Join Algorithm In MapReduce. *CoRR*, abs/1410.4156, 2014.
- [7] M. Aigner and M. Fromme. Game of cops and robbers. *Discrete Applied Mathematics*, 8(1), pp. 1–12, 1984.
- [8] N. Alon and A. Mehrabian. Chasing a Fast Robber on Planar Graphs and Random Graphs. *Journal of Graph Theory*, 78(2), pp. 81–96, 2015.
- [9] K. Amroun, Z. Habbas, and W. Aggoune-Mtala. A compressed Generalized Hypertree Decomposition-based solving technique for non-binary Constraint Satisfaction Problems. *AI Communications*, 29(2), pp. 371–392, 2016.
- [10] W.D. Baird. Cops and Robbers on Graphs and Hypergraphs. Ryerson University, Theses and dissertations. Paper 821, 2011.
- [11] C. Beeri, R. Fagin, D. Maier, and M. Yannakakis. On the Desirability of Acyclic Database Schemes. *Journal of the ACM*, 30(3), pp. 479–513, 1983.
- [12] P.A. Bernstein and N. Goodman. The power of natural semijoins. *SIAM Journal on Computing*, 10(4), pp. 751–771, 1981.
- [13] H.L. Bodlaender and F.V. Fomin. A Linear-Time Algorithm for Finding Tree-Decompositions of Small Treewidth. *SIAM Journal on Computing*, 25(6), pp. 1305–1317, 1996.
- [14] A. Bonato and R.J. Nowakowski. The game of cops and robbers on graphs. *Student Mathematical Library*, vol. 61, American Mathematical Society.
- [15] A.K. Chandra, D.C. Kozen, and L.J. Stockmeyer. Alternation. *Journal of the ACM*, 26:114–133, 1981.

- [16] D. A. Cohen and P. Jeavons. The Complexity of Constraint Languages. In *Handbook of Constraint Programming*, F. Rossi, P. van Beek, and T. Walsh, Eds., Elsevier, 2006.
- [17] D. A. Cohen, P. Jeavons, and M. Gyssens. A unified theory of structural tractability for constraint satisfaction problems. *Journal of Computer and System Sciences*, 74(5):721–743, 2008.
- [18] M.C. Cooper and S. Zivny. Hybrid tractability of valued constraint problems. *Artificial Intelligence*, 175(9-10):1555–1569, 2011.
- [19] J. Dean and S. Ghemawat. MapReduce: a flexible data processing tool. *Communications of the ACM*, 53(1):72–77, 2010.
- [20] D. Dereniowska, D. Dyerb, R.M. Tifembachc, B. Yangd. The complexity of zero-visibility cops and robber. *Theoretical Computer Science*, 607(2), pp. 135–148, 2015.
- [21] R. Dechter. Constraint Processing. Morgan Kaufmann, 2003.
- [22] R. Fagin. Degrees of acyclicity for hypergraphs and relational database schemes. *Journal of the ACM*, 30(3):514–550, 1983.
- [23] F.V. Fomin, P.A. Golovach, J. Kratochvíl, N. Nisse, and K. Suchan. Pursuing a fast robber on a graph. *Theoretical Computer Science*, 411(7-9), pp. 1167–1181, 2010.
- [24] F.V. Fomin, P.A. Golovach, and P. Pralat. Cops and Robber with constraints. *SIAM Journal on Discrete Mathematics*, 26, pp. 571–590, 2012.
- [25] P. Frankl. Cops and robbers in graphs with large girth and Cayley graphs. *Discrete Applied Mathematics*, 17(1), pp. 301–305, 1987.
- [26] A. Frieze, M. Krivelevich, and P. Loh. Variations on cops and robbers. *Journal of Graph Theory*, 69(4), pp. 383–402, 2012.
- [27] T. Gavenciak, V. Jelínek, P. Klavík, and J. Kratochvíl. Cops and Robbers on Intersection Graphs. In *Proc. of ISAAC’13*, pp. 174–184, 2013.
- [28] L. Ghionna, G. Greco, and F. Scarcello. H-DB: A Hybrid Quantitative-structural SQL Optimizer. In *Proc. of CIKM ’11*, pp. 2573–2576, 2011.
- [29] L. Ghionna, and L. Granata, G. Greco, and F. Scarcello. Hypertree Decompositions for Query Optimization. In *Proc. of ICDE’07*, pp. 36–45, 2007.
- [30] A.S. Goldstein and E.M. Reingold. The complexity of pursuit on a graph. *Theoretical Computer Science*, 143(1), pp. 93–112, 1995.
- [31] G. Gottlob, G. Greco, and F. Scarcello. Treewidth and hypertree width. In *Tractability: Practical Approaches to Hard Problems*, Cambridge Press, 2014.
- [32] G. Gottlob, G. Greco, N. Leone, and F. Scarcello. Hypertree Decompositions: Questions and Answers. In *Proc. of PODS’16*, San Francisco, CA, USA, 2016.
- [33] G. Gottlob, N. Leone, and F. Scarcello. The complexity of acyclic conjunctive queries. *Journal of the ACM*, 48(3), pp. 431–498, 2001.
- [34] G. Gottlob, N. Leone, and F. Scarcello. Hypertree decompositions and tractable queries. *Journal of Computer and System Sciences*, 64(3), pp. 579–627, 2002.
- [35] G. Gottlob, N. Leone, and F. Scarcello. Robbers, marshals, and guards: game theoretic and logical characterizations of hypertree width. *Journal of Computer and System Sciences*, 66(4), pp. 775–808, 2003.
- [36] G. Gottlob, Z. Miklós, and T. Schwentick. Generalized hypertree decompositions: NP-hardness and tractable variants. *Journal of the ACM*, 56(6), 2009.
- [37] N. Goodman and O. Shmueli. The tree projection theorem and relational query processing. *Journal of Computer and System Sciences*, 29(3), pp. 767–786, 1984.
- [38] G. Greco and F. Scarcello. Tree projections and structural decomposition methods: Minimality and game-theoretic characterization. *Theoretical Computer Science*, pp. 95–114, 2014.
- [39] G. Greco and F. Scarcello. On the power of structural decompositions of graph-based representations of constraint problems. *Artificial Intelligence*, 174(5-6):382–409, 2010.
- [40] G. Greco and F. Scarcello. The Power of Tree Projections: Local Consistency, Greedy Algorithms, and Larger Islands of Tractability. In *Proc. of PODS’10*, pp. 327–338, 2010.
- [41] G. Greco and F. Scarcello. Counting solutions to conjunctive queries: structural and hybrid tractability. In *Proc. of PODS’14*, pp. 132–143, 2014.
- [42] M. Grohe. The complexity of homomorphism and constraint satisfaction problems seen from the other side. *Journal of the ACM*, 54(1), 2007.
- [43] M. Grohe and D. Marx. Constraint solving via fractional edge covers. *ACM Transactions on Algorithms*, 4:1–20, 2014.
- [44] D.S. Johnson. A Catalog of Complexity Classes, *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity*, pp. 67–161, 1990.
- [45] O. Kalinsky, Y. Etsion, and B. Kimelfeld. Flexible Caching in Trie Joins. *CoRR*, abs/1602.08721,

- 2016.
- [46] W.B. Kinnersley. Cops and Robbers is EXPTIME-complete. *Journal of Combinatorial Theory, Series B*, 111, pp. 201–220, 2015.
 - [47] Ph.G. Kolaitis. Constraint Satisfaction, Databases, and Logic. In *Proc. of IJCAI'03*, pp. 1587–1595, 2003.
 - [48] M. Mamino. On the computational complexity of a game of cops and robbers. *Theoretical Computer Science*, 477, pp. 48–56, 2013.
 - [49] D. Marx. Approximating fractional hypertree width. *ACM Transactions on Algorithms*, 6(2), 2010.
 - [50] D. Marx. Tractable Hypergraph Properties for Constraint Satisfaction and Conjunctive Queries. *Journal of the ACM*, 60(6), 2013.
 - [51] R. Nowakowski and P. Winkler. Vertex-to-vertex pursuit in a graph. *Discrete Mathematics*, 43(2-3), pp. 235–239, 1983.
 - [52] A. Quilliot. Discrete pursuit game. *Congr. Numer.*, 38, pp. 227–241, 1983.
 - [53] N. Robertson and P.D. Seymour. Graph minors III: Planar tree-width. *Journal of Combinatorial Theory, Series B*, 36, pp. 49–64, 1984.
 - [54] P.D. Seymour and R. Thomas. Graph searching and a min-max theorem for tree-width. *Journal of Combinatorial Theory, Series B*, 58, pp. 22–33, 1993.
 - [55] B.S.W. Schroeder. The copnumber of a graph is bounded by $\lfloor 3/2 \text{genus}(g) \rfloor + 3$, *Trends in Mathematics*, pp. 243–263, 2001.
 - [56] J. D. Ullman. *Principles of Database and Knowledge Base Systems*. Computer Science Press, 1989.
 - [57] M. Yannakakis. Algorithms for acyclic database schemes. In *Proc. of VLDB'81*, pp. 82–94, 1981.